Handelshögskolan
VID GÖTEBORGS UNIVERSITET
Institutionen för informatik

2005-05-27

KNAPPTRYCKNING ELLER TIDSKRÄVANDE PROCESS?
Om problem vid migrering för plattformsoberoende

Abstrakt

Ett ökat intresse för alternativ till marknadsledande operativsystem och annan mjukvara har skapat ett behov hos mjukvaruleverantörer av att i konkurrenssyfte kunna utöka sin målgrupp genom att göra applikationer tillgängliga för användare av ett flertal plattformar. Att migrera en befintlig applikation till en annan plattform är fördelaktigt, inte bara för kostnadseffektiviteten, utan även ur kvalitetsaspekt. I denna studie har ett tre månader långt experiment utförts på ett företag som saluförde en applikation utvecklad för Windows och hade för avsikt att göra den tillgänglig för en större kundkrets. Experimentet utgjordes av en migrering av denna plattformsspecifika Windows-applikation för att göra den tillgänglig för användare av andra operativsystem, såsom Linux och Mac OS. Experimentet resulterade i fältanteckningar som efter sammanställning och analys kunde belysa problem som kan uppstå vid en migrering. Resultatet utgjordes av ett flertal problem som relaterade till det faktum att applikationen migrerades mellan plattformar. Det som ytterligare komplicerade processen var strävan att utveckla en, och endast en, uppsättning källkod för applikationen, vilken kunde exekveras på ett flertal plattformar. Resultatet låter se rekommendation att lägga stor vikt vid förståelsen för den ursprungliga applikationen, dess källkod och funktionalitet. De initiala stadierna i migreringen tillsammans med hantering av externa bibliotek visade sig vara de mest tidskritiska.

Nyckelord: Migrering, portning, plattformsoberoende, Windows, Linux, Mac, Java, C++

Författare: Lina Larsson, Petra Kyrkander
Handledare: Johan Magnusson
Magisteruppsats, 20 poäng

Postadress	*Besöksadress*	*Telefon*	*Telefax*
Handelshögskolan vid Göteborgs universitet Institutionen för informatik Box 620 SE 405 30 GÖTEBORG	Viktoriagatan 13	031- 773 10 00	031 - 773 47 54

Förord

Vi vill rikta ett stort tack till vår handledare Johan Magnusson, som uppmuntrat, inspirerat och tålmodigt besvarat alla våra frågor under uppsatsskrivningen.

Vi vill också tacka samtliga anställda på företaget där experimentet utfördes för all hjälp vi fått under vistelsen på företaget. Speciellt tack till Peter som givit oss möjlighet och resurser för att undersöka vårt problemområde i praktiken, Björn som givit oss stöd under hela experimentet samt Anders, en aldrig sinande inspirationskälla som trots ett pressat arbetsschema alltid tagit sig tid att hjälpa oss igenom källkodens labyrinter.

Ett stort tack vill vi också rikta till familj och vänner, som har stått ut med oss, och dessutom stöttat och uppmuntrat oss, under den här perioden.

Tack till alla er som på ett eller annat sätt gjort det möjligt för oss att slutföra vår magisteruppsats.

Innehållsförteckning

Figurförteckning

Tabellförteckning

Introduktion

Att migrera en befintlig applikation för att göra den tillgänglig för en större marknad har en mängd fördelar för de företag som saluför produkten. Den introduktionsprocess som innebär att sammanställa och utveckla en idé som kan anses vara säljande på marknaden, för att sedan marknadsföra den och utreda marknadens intresse, är redan utförd.[1] Man har redan tillgång till en produkt som har visat sig intressant på marknaden. En migrering till plattformsoberoende innebär tillgång till en utökad målgrupp, då användare av flertalet operativsystem kan utnyttja den, vilket leder till konkurrensfördelar.[2] Migreringsprocessen innebär inte bara utvecklandet av produkten för nya plattformar, utan även en revidering av den befintliga produkten, vilket troligen leder till en kvalitetsförbättring även för den ursprungliga produkten. Även design- och övriga dokument revideras och en korrekt utförd migrering kommer också leda till en, så långt möjligt, gemensam källkodsbas, vilket kommer att underlätta framtida underhåll av både ursprunglig och migrerad produkt.[3] För att sammanfatta leder migrering i många fall till:

- Konkurrensfördelar – genom en utökad målgrupp
- Kvalitet – genom revidering av källkoden
- Förenklad underhållning – genom förbättrad dokumentation och gemensam källkodsbas

Om själva migreringen blir en enkel eller komplicerad process, är beroende av ett flertal faktorer. För att få en bild av vilka problem som kan förekomma vid en migrering har ett tre månader långt experiment utförts. Experimentet i fråga bestod i att migrera en befintlig applikation ursprungligen utvecklad för Windows i syfte att ge den plattformsoberoende egenskaper. Den ursprungliga applikationens källkod, inklusive gränssnitt, är utvecklad i programspråket C++[4] och utnyttjar biblioteket Microsoft Foundation Class Library (MFC).[5] Applikationen har specifika funktioner i form av kryptering, vilken utförs med hjälp av ytterligare två bibliotek, nämligen Synchronous Key Generator (SKG) samt Crypto++.

För att uppnå syftet med experimentet avgjordes att utvecklingen skulle ske i Java, just på grund av språkets plattformsoberoende egenskaper. Uppgiften bestod då i att migrera den befintliga produkten och i processen finna komponenter som var kompatibla med det nya programspråket och de nya plattformarna, samt gav möjlighet att behålla ursprunglig funktionalitet.

Här ges en introduktion till det område som kommer att behandlas i uppsatsen. En beskrivning av problemområdet som en företeelse av intresse att studera följs av syftet, avsnittet som i korthet beskriver vad uppsatsen kommer att behandla. Detta avsnitt följs av en beskrivning av vilka läsare uppsatsen riktar sig till. Därefter påvisas avgränsningar, för att tydliggöra vilka områden som ligger utanför uppsatsens ramar och därför inte kommer att behandlas.

---

[1] Cravotta, Robert, *Fear and loathing of porting embedded software,* 2002.
[2] Mindfiresolutions, *Porting: The business imperative*, 2001.
[3] Mindfiresolutions, *Porting: A development primer*, 2001.
[4] Definition av *"C++"*, se Bilaga 1.
[5] Definition av *"Microsoft Foundation Class Library"*, se Bilaga 1.

Problemområde

Datorer säljs nu för tiden vanligtvis med ett förinstallerat operativsystem vars syfte är att organisera och kontrollera datorns hårdvara och mjukvara. De mest förekommande operativsystemen idag är Windows-familjen som utvecklas av Microsoft, Mac OS-familjen som utvecklas av Apple samt Unix och Linux som utvecklas av privatpersoner och företag.[6] Fördelningen mellan dessa operativsystem ser ut på följande sätt: ca 90 % Windows-användare, ca 3 % Mac-användare och ca 3 % Linux-användare.[7] Dessa siffror baserar sig på Internetanvändares val av operativsystem.

Enligt forsknings- och analysföretaget Gartner försöker många statliga verksamheter och företag världen över skydda sig från Microsofts växande inflytande på IT-industrin. Linux och annan mjukvara med öppen källkod har därför blivit intressanta alternativ.[8] Artiklar visar att små- och mellanstora företag börjar intressera sig för Mac, och en kombination av en kraftig prisreduktion på Macintoshdatorer samt Apples försäljningssuccé Ipod har gjort att även privatpersoner börjat intressera sig för Mac och Mac OS.[9,10]

Även om de flesta datoranvändare använder Windows har företag som har en befintlig operativsystemspecifik, eller plattformsspecifik produkt, mycket att vinna genom att göra produkten plattformsoberoende. Dels kan man utöka sin målgrupp, det vill säga, man ger fler användare möjlighet att utnyttja produkten, vilket ökar företagets konkurrenskraftighet. Att göra produkten tillgänglig för användare av en annan plattform kan även öka det allmänna intresset för produkten och kan på så sätt, eller genom effekter av marknadsföring för den nya produkten, också öka intresset för den tidigare utgåvan. Revidering av produktens källkod vid en migrering kan dessutom resultera i kvalitetsförbättringar, även för den ursprungliga produkten.[11]

Det finns en mängd information om migrering mellan olika plattformar och programspråk att finna. Utvecklare av operativsystem delar mer än gärna med sig av information som berör hur man migrerar applikationer för att kunna exekvera dem på deras plattform. Av förklarliga skäl är de mer restriktiva med information som ger stöd vid migrering för plattformsoberoende, vilket resulterar i att denna typ av information är svårare att finna.

För att utveckla en plattformsoberoende applikation krävs ett programmeringsspråk vilket kan kompileras och exekveras på olika operativsystem. Många programmeringsspråk har denna egenskap. Java är ett av dem. Den första utgåvan av Java kom år 1995 och har sedan dess vuxit till att bli ett av de mest använda programmeringsspråken. Det som gör Java speciellt är att källkoden exekveras mot en virtuell maskin och inte direkt mot operativsystemet. Detta har gjort det möjligt att utveckla programvara för olika plattformar utan att programmeraren nödvändigtvis behöver skriva eller kompilera plattformsspecifik kod.[12]

Java har blivit vida accepterat som ett språk med vilket man kan skapa plattformsoberoende applikationer,[13] och det finns ett antal verktyg och böcker till hjälp för att migrera Windows-

[6] Coustan, Dave och Curt Franklin, *How Operating Systems Work.*
[7] W3Schools, *Browser Statistics.*
[8] Smith, David Mitchell och Robin Simpson. *A Look at Alternatives to Microsoft*, 2003.
[9] Best, Jo, *Survey: Some iPod fans dump PCs for Macs*.
[10] Bohlin, Stefan. ”All vill sälja Ipod och Mac”, Computer Sweden Nr40 (2005): 9.
[11] Mindfire solutions. *Porting: The Business Imperative*.
[12] Cadenhead, Rogers och Laura Lemay, *Sams Lär dig Java 2 på 3 veckor*. (Falun: Pagina Förlags AB, 2003).
[13] Modi, Tarak, *The Platform-Blending Concept*, 2000.

applikationer skrivna i programmeringsspråket C++ till Java. Det är dock svårt att hitta någon forskning eller dokumentation förutom i forum för programmering som dokumenterar de övergripande problem som kan uppstå vid migrering från en Windows-applikation till en plattformsoberoende applikation.

Syfte

Syftet med denna uppsats är att kartlägga och beskriva problem som kan uppstå då en Windows-applikation migreras för att göras plattformsoberoende.

Målgrupp

Denna uppsats riktar sig framför allt till de läsare som har kunskap inom området och eventuellt har stött på problem av liknande karaktär. Antagandet om läsarens kännedom om här berörda områden gör att fackspecifika ord som förekommer saknar beskrivning i texten. För den som saknar definitioner av dessa finns dock utvalda ord beskrivna i Bilaga 1.[14]

Avgränsningar

Syftet med uppsatsen är belysa problem som kan uppstå vid migrering för att uppnå plattformsoberoende. Syftet är därför inte huvudsakligen att framhäva egenskaperna hos olika operativsystem, programspråk eller programspråkens tillhörande bibliotek. Dessa kommer därför diskuteras endast i den omfattning som är nödvändig för en allmän förståelse för uppsatsen som helhet. Uppsatsen berör endast den del av migreringsprocessen som behandlar översättning av källkod och problem med skillnader mellan plattformar. Därför lämnas testningsprocessen utanför uppsatsens ram. I de flesta fall när man talar om migrering mellan plattformar, är det ett operativsystem som avses, men migrering kan även röra sig om att växla databaser eller kompilatorer.[15] Här är det dock migrering mellan operativsystem som åsyftas. De operativsystem som innefattas i uppsatsens definition av plattformsoberoende har begränsats till Windows, Mac OS och Linux.

Den produkt som var mål för experimentet har utvecklats med fokus på säkerhet och integritet. Detta gjorde att experimentet till stor del präglades av säkerhetsaspekter och faktorer rörande kryptering. Dessa aspekter utgör ett omfattande område, som dock här kommer att nämnas förhållandevis lite, eftersom de i sig faller utanför ramen för de diskussioner som är ämnade leda till uppfyllande av syftet för uppsatsen.

[14] Definitioner, se Bilaga 1.
[15] Kelley, Allan, *Where to begin: Porting,* 2001.

Disposition

Uppsatsen är uppdelad i 5 kapitel: introduktion, metod, teori, resultat och diskussion och generell diskussion. Nedan beskrivs kortfattat varje kapitel.

I introduktionen förklaras bakgrunden till denna uppsats, varför problematiken är intressant, och syftet med uppsatsen. Introduktionen ger även en inblick i det experiment som utförts för att kunna uppnå uppsatsens syfte samt avgränsningar där det beskrivs vad, inom uppsatsens problemområde, uppsatsen inte behandlar.

Metodkapitlet beskriver de metoder som använts, varför de har valts och vilka synsätt som påverkat uppsatsen. Kapitlet består även av en beskrivning av det praktiska tillvägagångssättet för experimentet som innefattar insamling och bearbetning av empiri.

Kapitlet teori är uppsatsens teoretiska fundament och beskriver således de teorier som ligger till grund för uppsatsen. I detta kapitel redogörs även för de problem och lösningar som uppkommit vid migreringar av applikationer.

I kapitlet resultat och diskussion presenteras och diskuteras de problem som stötts på under migreringsexperimentet. För att göra avsnittet överskådligt har problemen kategoriserats utefter deras karaktär.

Det avslutande kapitlet innehåller även det en diskussion. Denna diskussion tar upp problemen som framkommit i uppsatsen ur ett generellt perspektiv.

Metod

Under denna rubrik presenteras tillvägagångssättet för att uppnå syftet med uppsatsen. Avsikten med metodavsnittet är att ge läsaren en möjlighet att kontrollera resultatet samt en möjlighet att göra en värdering av det empiriska förfarandet.

Avsnittet börjar med beskrivningar av de metoder som använts för att utföra experimentet samt för att samla in och bearbeta empiri. Beskrivning sker först ur teoretiskt, sedan ur praktiskt perspektiv. Därefter följer en redogörelse för den utförda litteraturstudien och en beskrivning av forskningskvalitet.

Vetenskapsteori och metodologi

Val av vetenskapsteori och metodologi

Inom informatikdisciplinen finns det idag många olika accepterade forskningsmetoder.[16] För att genomföra detta experiment och nå ett resultat behövdes en metod där observation och systemutveckling kunde utföras parallellt. Litteraturstudien av metodik gav inblick i olika metoder som skulle gå att kombinera med de krav som fanns. Innan en forskningsmetod väljs är det viktigt att vara medveten om vilka eventuella fördelar och nackdelar metoden för med sig samt i vilka sammanhang metoden är lämplig att använda.[17] Efter en granskning av för- och nackdelar med de olika metoderna beslutades att den metod som bäst passade som övergripande experimentmetod var en forskningsmetod kallad Concurrent ethnography. För att samla in data, har som huvudmetod valts deltagande observation enligt Concurrent ethnography med fältanteckningar enligt Grounded theory. Sammanställning och analys av data har gjorts med hjälp av Grounded theory och hyptotetiskt-deduktiv metod.

Nedan följer en närmare beskrivning av de olika metoder och angreppssätt som tillämpats för att nå resultat.

Kvalitativ och kvantitativ

Kvantitativa är metoder som leder till numeriska observationer eller låter sig transformeras i sådana. De metoder som istället resulterar i verbala formuleringar kallas kvalitativa.[18] Att använda sig av en kvantitativ metod innebär att man förutsätter tänkbara resultat, vilket man inte gör med en kvalitativ metod som tar hänsyn till helheten på ett helt annat sätt. Medan den kvantitativa metoden strävar efter att förklara ett förutsatt fenomen, vill man istället med den kvalitativa förstå ett fenomen. Man har en öppenhet inför resultatet, och kräver följsamhet mot data, det vill säga tillvägagångssättet för insamling och tolkning kan behöva förändras under arbetets gång. Syftet med insamlad data är att beskriva essenser, teman, mönster och uppfattningar.[19] Beskrivningen som görs är beroende av vilka frågor man vill besvara och omfattar vanligtvis en hel aktivitet för att möjliggöra en holistisk bild av det undersökta.[20]

---

[16] Myers, M. D., *Investigation information systems with ethnographic research*. Auckland: University of Auckland, 1999.
[17] Ibid.
[18] Backman, Jarl, *Rapporter och uppsatser*, (Lund: Studentlitteratur, 1998).
[19] Gunnarsson, Ronny, *Forskningsansats – kvalitativt eller kvantitativt perspektiv?*
[20] Genzuk, M., *A synthesis of ethnographic research*. Southern California: University of Southern California Center for Multilingual, Multicultural Research.

Etnografisk metod

Etnografin är en metod som ursprungligen utvecklades av antropologer för att förstå sociala mekanismer i primitiva samhällen. Den involverade antropologer som levde några år i ett samhälle och gjorde detaljerade observationer. Analysen av dessa observationer resulterade i information om samhällets struktur och organisering.[21]

Etnografi karaktäriseras bland annat av att:

- det inte finns några antaganden om objektet som studeras och att etnografen försöker vara så objektiv som möjligt.[22]
- data samlas in från en mängd olika källor, men data från observationer är ofta huvudkällan.[23]
- tillvägagångssättet vid insamlande av data är ostrukturerad i den bemärkelse att det inte innebär att forskaren har en detaljerad plan från början. Med detta menas inte att forskningen är osystematisk; bara att det till en början är rådata som samlas in.[24]
- analysen av data, som ofta tar formen av verbala beskrivningar och förklaringar, innebär tolkning av rådata.[25]

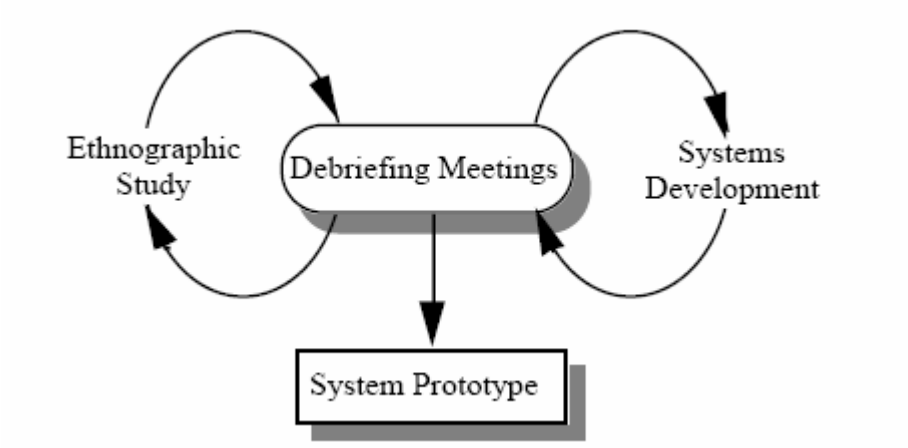
Inom informatikämnet används etnografisk undersökningsmetod bland annat för att studera och analysera verksamheter. Detta sker genom att en etnograf observerar flöden, processer och samarbete inom en organisation. Anledningen till dessa studier är att arbetsmomenten ofta skiljer sig markant från det som forskaren kan få förmedlat genom att intervjua de anställda eller via företagets arbetsmanualer. Resultatet av denna typ av etnografiska studier kan resultera i en kravspecifikation för ett informationssystem.[26]

Concurrent ethnography

Hughes har specificerat fyra förslag att använda etnografin inom systemdesign – Concurrent ethnography, Quick and dirty ethnography, Evaluative ethnography samt Re-examination of wevious studies.[27] Concurrent ethnography är förmodligen den etnografiska metod som vanligtvis associeras med systemdesign. Metoden passade för experimentet som skulle utföras eftersom processen för metoden är sekventiell och iterativ och innebär att en etnografisk studie görs vilken efterföljs av ett möte mellan etnografer och systemutvecklare och efter detta utförs systemdesign. Dessa steg repeteras till dess att resultatet är tillfredsställande.[28]

---

[21] Sommerville, I. et al, Integrating ethnography into the requirements engineering process. Lancaster: Lancaster University, 1994.
[22] Ibid.
[23] Genzuk, M., *A synthesis of ethnographic research*.
[24] Ibid.
[25] Ibid.
[26] Sommerville, I. et al, *Integrating ethnography into the requirements engineering process*.
[27] Hughes, J., *Moving Out from the Control Room: Ethnography in System Design*. Lancaster: Lancaster University, 1994.
[28] Ibid.



Figur 1 Concurrent Ethnography[29]

Forskningsmetodens observationskaraktär gjorde att ett beslut om graden av involvering i de observerande aktiviteterna var nödvändigt att fatta.[30,31] De två ytterligheterna är fullständig separation från det observerade objektet och fullständigt engagemang i objektet som ska observeras. Ytterligheterna kallas inom forskningen för icke-deltagande observation respektive deltagande observation. Mellan dessa ytterligheter finns en stor variation av olika förhållningssätt till det studerade objektet.[32]

Deltagande observation är en allomfattande strategi som inbegriper intervjuer, direkt deltagande och observation samt själviakttagelse. Forskaren gör sig själv till en del av det studerade objektet, detta för att få en större förståelse för fenomenet som studeras. Detta medför att forskaren får två roller – en deltagande och en observerande och måste därmed kombinera deltagande och observation så att han eller hon förstår upplevelsen som deltagande men beskriver den objektivt.[33] Eftersom forskaren får en förståelse för objektet som studeras genom interaktion men även genom observation ansågs deltagande observation lämplig för experimentet som skulle utföras.

Graden av involvering i experimentet som utfördes motsvarades av ett fullständigt deltagande i det studerade objektet. Experimentet utfördes och observerades utifrån i samma skede. Vid observation rekommenderas olika tekniker för datainsamling.[34] Den teknik som var mest intressant i denna undersökning var att ta fältanteckningar som beskrev hur experimentet fortskred och som metod för detta användes Grounded theory som står att läsa om längre fram i detta metodavsnitt.

Då man använder sig av en etnografisk metod och har en infallsvinkel som innebär deltagande observation, finns en del kritik man bör ta hänsyn till.[35,36] Kritik som riktats mot den etnografiska undersökningsmetoden är bland annat att den inte har någon stor bredd. Forskaren inriktar sig endast på en situation eller företeelse. Några forskare anser att studien därför leder endast till en djup förståelse för den specifika situationen och att det därför inte

[29] Hughes, J., *Moving Out from the Control Room: Ethnography in System Design*.
[30] Frechtling, Joy and Laure Sharp, *User-Friendly Handbook for Mixed Method Evaluations*, 1997.
[31] Ibid.
[32] Genzuk, M., *A synthesis of ethnographic research*.
[33] Ibid.
34 Hancock, Beverley, *An Introduction to Qualitative Research*, University of Nottingham, 1998.
[35] Genzuk, M., *A synthesis of ethnographic research*.
[36] Hancock, Beverley, *An Introduction to Qualitative Research.*

går att göra generella modeller utifrån endast en etnografisk studie. Andra forskare menar att det visst går att generalisera utifrån endast en undersökning.[37] Detta har tagits hänsyn till i denna studie och kritiken bemöts under rubrikerna Validitet och reliabilitet samt under Objektivitet längre fram i metodavsnittet.

Grounded theory

Grounded theory är en metodologi som utvecklades av Barney Glaser och Anselm Strauss.[38] Metoden syftar till att en teori ska skapas genom att systematiskt samla och analysera data där datainsamling och analys står nära relation till varandra.[39] Teorin som är målet vid användandet av metoden ska bestå av välutvecklade koncept eller grupperingar som tillsammans tillhandahåller ett ramverk som används för att förklara eller förutse ett fenomen. Teorin ska vara generaliserad så att den kan appliceras inte bara på det objekt som studerats.[40] Inom Grounded theory har bland annat föreslagits ett antal procedurer och tekniker som kan användas vid gruppering av data, samt definiering och utvecklande av kategorier.[41]

Grounded theory har främst använts inom sociologi, folkhälsovetenskap och omvårdnadsforskning, men har på senare tid börjat användas allt mer inom informatikdisciplinen.[42,43]

Induktion, deduktion och hypotetisk-deduktiv metod

Induktion och deduktion är två olika sätt att dra slutsatser på, där induktion bygger på empiri och deduktion bygger på logik.

Induktion innebär att generella slutsatser dras utifrån empiriska fakta och bygger oftast på ett antal observationer som tillsammans leder till en generalisering. [44,45] Sohlberg tar upp ett exempel på en induktiv slutsats:

”Solen har observerats vid horisonten varje dag hitintills och vi kan sluta oss till att den skall finnas där imorgon också”.[46]

En induktiv slutledning kan vara mer eller mindre sannolik, men aldrig hundraprocentigt säker eftersom den bygger på empiriskt material.[47] Det som är sant idag kan vara annorlunda imorgon, induktiva slutledningar är därför aldrig logiskt bindande.[48]

Deduktion innebär att en logisk slutsats dras, som betraktas som giltig om den är logiskt sammanhängande. Däremot behöver den inte vara sann i den mening att den stämmer överens med verkligheten. I deduktion tas inte hänsyn till om de grundläggande premisserna som

[37] Genzuk, M., *A synthesis of ethnographic research*.
[38] Strauss, Anselm och Juliet Corbin, *Basic qualitative research: Techniques and procedures for developing grounded theory*. (USA: Sage Publications, 1998).
[39] Ibid.
[40] Ibid.
[41] Ibid.
[42] Gunnarsson, Ronny, *Grounded theory*.
[43] Myers, Michael, *References on Grounded Theory.*
[44] Thurén, Torsten, *Vetenskapsteori för nybörjare* (Malmö: Liber AB, 1991).
[45] Sohlberg, Peter och Britt-Marie Sohlberg, *Kunskapens former – Vetenskapsteori och forskningsmetod* (Falköping: Liber AB, 2001).
[46] Ibid.
[47] Thurén, Torsten, *Vetenskapsteori för nybörjare*.
[48] Sohlberg, Peter och Britt-Marie Sohlberg, *Kunskapens former – Vetenskapsteori och forskningsmetod.*

ingår är sanna eller inte[49] och slutledningarna tillför inget nytt vad gäller substansen. Thurén visar på deduktiva slutsatser:[50]

Premiss 1*: Alla människor är dödliga*
Premiss 2*: Jag är människa*

Slutsats*: Alltså är jag dödlig*

Men även denna slutsats skulle vara deduktivt giltig:

Premiss 1*: Alla människor har fyra armar*
Premiss 2*: Jag är människa*

Slutsats*: Alltså har jag fyra armar*

Eftersom induktion bygger på empiriska data är det oftast det slutledningssättet som används inom kvalitativ forskning. Strauss och Corbin menar att även om relationer och hypoteser uppstår från data så tolkas data till en viss grad och detta betraktar de som en form av deduktion. Premisserna kan vara något otydliga men består av den empiri och de antaganden och tidigare kunskaper som finns, även om ett objektivt synsätt ska användas. Vidare menar de att det finns ett samspel mellan induktion och deduktion och detta samspel ska tas hänsyn till vid användandet av Grounded theory.[51] Detta sätt att se på det går i linje med vad Thurén kallar för hypotetiskt-deduktiv metod. Hypotetiskt-deduktiv metod innebär att man som premisser ställer upp hypoteser genom vilka det görs en deduktiv slutledning. Efter det kontrolleras om premisserna som ställts upp stämmer med verkligheten. Genom denna metod kombineras både induktion och deduktion och därmed empiri och logik.[52] Thurén visar på ett exempel:[53]

Premiss 1 (hypotes)*: om dödligheten beror på att studenterna för med sig giftiga ämnen från obduktionerna så kommer dödligheten att minska om dessa ämnen avlägsnas.*
Premiss 2*: Om studenterna tvättar händerna i klorvatten kommer dessa ämnen att avlägsnas.*
Premiss 3 (observation)*: När studenterna tvättar händerna minskar dödligheten.*

Slutsats*: Alltså beror skillnaden i dödlighet på giftiga ämnen från obduktionen.*

Till denna studie har hypotetiskt-deduktiv metod använts för att göra slutledningar. Detta har inburit att för varje problem har framtagits troliga hypoteser kring varför problemet uppstått

[49] Thurén, Torsten, *Vetenskapsteori för nybörjare*.
[50] Thurén, Torsten, *Vetenskapsteori för nybörjare,* 23, 24.
[51] Strauss, Anselm och Juliet Corbin, *Basic qualitative research: Techniques and procedures for developing grounded theory*.
[52] Thurén, Torsten, *Vetenskapsteori för nybörjare.*
[53] Thurén, Torsten, *Vetenskapsteori för nybörjare*, 27.

och hur en eventuell lösning skulle se ut. Varje hypotes har därefter med hjälp av observationer antigen verifierats eller falsifierats. En mer ingående förklaring av metodens inverkan finns att läsa om i under rubriken Praktiskt genomförande.

Validitet och reliabilitet

Validitet kan delas upp i två nivåer: intern och extern validitet. Intern validitet behandlar frågan om i vilken mån ens resultat stämmer överens med verkligheten. Extern validitet innebär den utsträckning i vilken resultaten är tillämpliga även i andra situationer än den undersökta. Reliabilitet handlar om i vilken utsträckning ens resultat kan upprepas.[54]

I kvalitativa studier som den här kan det vara svårt att följa upp validiteten och reliabiliteten. Då en kvalitativ studies eller ett kvalitativt experiments validitet och reliabilitet ska undersökas är det viktigt att titta på de instrument som använts. Det är också viktigt att se till lämpligheten av att använda de analystekniker som valts eller till relationerna som finns mellan de slutsatser som dragits och den informationen som utgåtts från. Oavsett vilken typ av forskning det handlar om är validitet och reliabilitet frågor som kan besvaras genom en noggrann uppmärksamhet på de grundläggande begreppen i undersökningen och på hur data samlats in, analyserats och tolkats.[55]

Eftersom termen reliabilitet i traditionell mening inte stämmer särskilt väl när den används på kvalitativ forskning är det viktigt att denna studie uppnår resultat som har någon mening och att de är konsistenta och beroende. Sharan Merriam föreslår ett antal tekniker som kan användas för att försäkra sig om att resultaten är beroende:[56]

TEKNIK	BESKRIVNING	GENOMSLAG I DENNA STUDIE
Forskarens position	Då en studie görs ska bakomliggande antaganden, metoder och teorier som ligger till grund för studien, och även den ställning som has till den grupp som studeras, förklaras.	Detta har uppfyllts genom introduktions- och metodkapitlet i denna uppsats.
Triangulering	Triangulering som innebär att olika metoder används för datainsamling stärker både den inre validiteten och reliabiliteten.	Datainsamling har gjorts med hjälp av Concurrent ethnography, deltagande observation samt fältanteckningsinsamling enligt Grounded Theory.
Transparens	En detaljerad beskrivning av hur informationen samlats in, hur kategoriseringen utvecklades samt vilka beslut som fattades under olika skeden i studien måste finnas. Studien som gjorts ska genom beskrivningen gå att replikera.	Detta uppfylls genom metodkapitlet i uppsatsen.

Tabell 1 Tekniker för att uppnå validitet och reliabilitet

[54] Merriam, B Sharan, *Fallstudien som forskningsmetod* (Lund: Studentlitteratur, 1994).
[55] Ibid.
[56] Ibid.

Då en metodologi som innebär att kontinuerlig datainsamling och analys används är det extra viktigt att objektivitet till materialet bibehålls under hela processen. Om detta görs kommer det resultat som tas fram att till högre grad vara opartiskt och tillförlitligt. Med att bibehålla objektivitet i studien menas att datainsamling, sammanställning och analys görs fria från egna antaganden, förutfattade meningar och erfarenheter. Föremålet för datainsamling och data i sig ska ses med ett öppet sinne.[57]

Strauss och Corbin menar att det i princip är omöjligt att vara helt objektiv, att det alltid finns ett element av subjektivitet inom både kvalitativ och kvantitativ forskning, men att det för att få ett bra resultat gäller att ifrågasätta allting och att inte ta något för givet. Nedan följer fem tekniker som Strauss och Corbin utvecklat, som ökar medvetandet om objektiviteten och ger möjlighet att kontrollera den.[58]

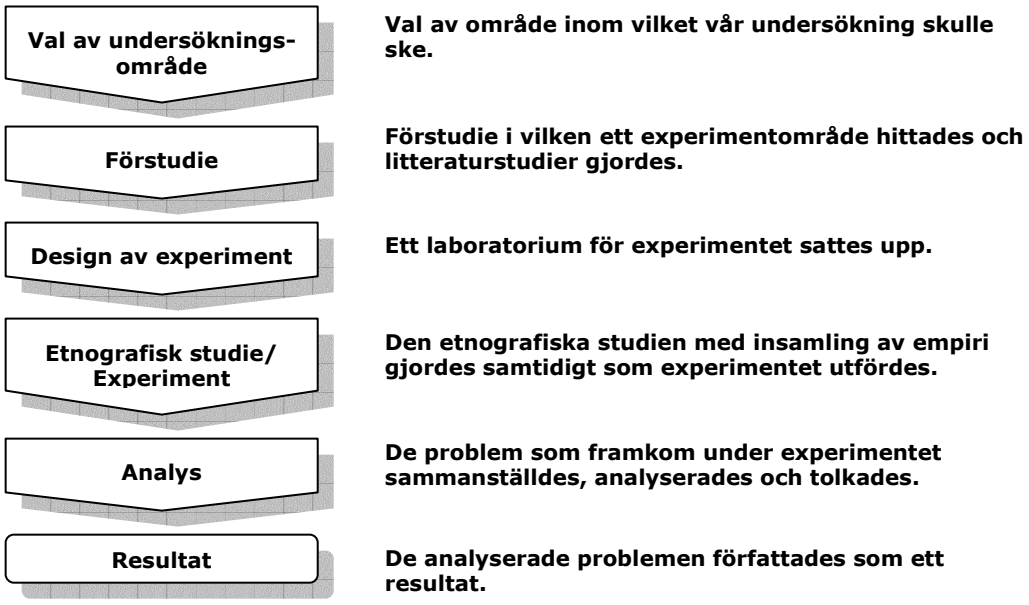
TEKNIK	BESKRIVNING	GENOMSLAG I DENNA STUDIE
Jämföra	Genom att jämföra olika händelser med varandra och med litteratur uppkommer likheter och skillnader vilket leder till eftertänksamhet vad gäller hänsynstagandet till egenskaper i de olika fallen.	I datainsamling och analys har jämförelse med litteratur och andra personers erfarenheter gjorts.
Se saker ur olika perspektiv	Vid datainsamling så som intervjuer och observationer ska saker ses ur olika synvinklar, det leder till bra triangulering. Även data ska ses och tolkas ur olika synvinklar så alla möjliga fall täcks in.	Data har i denna studie studerats ur både ett praktiskt och ett teoretiskt perspektiv.
Ta ett steg tillbaka	Upprepade gånger under en studie är det viktigt att ta ett steg tillbaka och kontrollera vad som sker och kontrollera att data som tagits fram överensstämmer med verkligheten.	Kontinuerligt under studie och skrivande har tagits en paus från experiment och skrivande för att läsa igenom, begrunda och kontrollera den teori och data som hittats.
Skepticism	Det är viktigt att vara skeptisk mot de data som samlats in och hela tiden validera dessa mot andra data.	Den data som samlats in har hela tiden verifierats eller falsifierats mot sakenlig och tillförlitlig litteratur och forum.
Följa forsknings-procedurer	Att följa en forskningsprocedur för datainsamling, sammanställning och analys är mycket viktigt för att uppnå ett användbart resultat.	Till denna studie har en mängd olika metoder, så som Concurrent ethnography, Grounded theory och hypotetiskt-deduktiv metod, använts för datainsamling, sammanställning och analys.

Tabell 2 Tekniker för att uppnå objektivitet

[57] Strauss, Anselm och Juliet Corbin, *Basic qualitative research: Techniques and procedures for developing grounded theory*.
[58] Ibid.

Praktiskt genomförande

Det övergripande tillvägagångssättet under uppsatsskrivande och experiment förklaras med nedanstående bild:



Figur 2 Övergripande tillvägagångssätt

Förstudie

Då denna uppsats är en avslutande del av Systemvetarprogrammet, har det under hösten pågått många diskussioner rörande examensarbete vilket då var nära förestående. Under en av dessa diskuterades möjligheten till ett migreringsexperiment på ett företag i Göteborg. Det visade sig vid kontakt med företaget att de kunde erbjuda möjlighet och resurser för att utföra experiment vars karaktär skulle vara passande för uppsatsen. Flera områden diskuterades som aktuella och fördelaktiga. På grund av den knappa tid, avgjort ca tre månader, som fanns att ägna åt experimentet valdes ett migreringsexperiment av, vad som förväntades, passande omfattning.

Innan experimentet påbörjades genomfördes en omfattande förstudie vilken resulterade i avgränsningar, en problemformulering, ett precist syfte samt val av forskningsmetoder.
Förstudien bestod först och främst i att studera litteratur inom vetenskapsteori och metodologi, detta för att få en bra grund att utgå ifrån. Då lämpliga metoder hittats påbörjades förstudien av experimentet. Denna bestod i att studera litteratur rörande migrering för att hitta infallsvinklar och migreringsmetoder. Därefter utfördes reverse engineering på den befintliga källkoden för få en övergripande struktur över mjukvarans uppbyggnad. Informationssökning utfördes för att få en klarare bild av uppgiften som väntade och efter detta validerades och fastställdes metoder som genomgripande skulle avspeglas i experiment och uppsats.

Insamling av empiri

Experimentet som utfördes skedde under en begränsad tidsperiod på tre månader, där experimentet med förstudien som underlag bestod av migrering av källkod. Experimentet genomfördes med Concurrent ethnography och deltagande observation som övergripande metod. Metoden inverkade på experimentets utförande genom att deltagande i experimentet samt fältanteckningar och undersökning genomfördes kontinuerligt och parallellt. Metoden

innebar att på samma gång inneha två roller, dels i form av experimentutövare, dels som observerande av experimentet som företeelse. För att med stöd i denna metod kunna generera ett önskat resultat lades vid utövandet av den sistnämnda rollen betydande vikt vid ett objektivt synsätt.

Fältanteckningar togs, i enlighet med Grounded theory, dels då sådana faktorer stöttes på som tvingade till avvikelser från migreringsprocessen, dels då problem uppstod som krävde en lösning för att experimentet skulle kunna fortgå och dels då lösningar hittades. De anteckningar som gjordes under experimentet sågs sedan som underlag för uppsatsens empiri.[59]

Sammanställning av empiri

Under den inledande analysfasen ska informationen göras enhetlig genom identifiering av de data som utgör grund för kategoriindelningen. Analysen innebär bland annat en utveckling av begreppsliga kategorier som kan hjälpa läsaren att tolka informationen.[60]

Sammanställningen av empiri gjordes enligt Grounded theory där det första steget i sammanställningen är konceptualisering, vilket innebär att hitta de data som är betydelsefulla i de insamlade data som finns tillgänglig. Efter detta ses till varje datas specifika egenskaper och karaktär vilket leder fram till kategorisering av de data som har liknande egenskaper.[61]

I sammanställningsfasen bröts först all insamlad data ner i distinkta migreringsproblem vilka döptes efter deras samlade egenskaper. Då samma problem dök upp två eller fler gånger sammanfogades dessa till ett problem. Då problemen identifierats och namngivits påbörjades kategoriseringen vilken syftar till att lyfta grupper av problem till en högre, mer abstrakt nivå.[62] Passande kategorinamn togs fram och därefter började grupperingen av problemen. De problemen med liknande egenskaper och karaktär grupperades under samma kategori och i de fall då ett problem var svårkategoriserat valdes den kategori där problemets egenskaper stämde bäst överens med de i kategorin redan existerande problemen.

Analys och tolkning av empiri

Analys av information är den process som går ut på att skapa mening av den. Målet för analysen är att komma fram till trovärdiga slutsatser och generaliseringar som har sin grund i empiriska data.[63] De empiriska data som insamlats lär inte bara om det specifika fallet utan även om andra liknande fall vilket leder till att det går att generalisera data. Generalisering av data kan göras genom jämförelse av liknande situationer. Genom att göra en sådan jämförelse kan både likheter och olikheter upptäckas.[64]

Analysen i denna studie har gjorts kontinuerligt och i enlighet med Grounded theory och Strauss och Corbins jämförelseteknik Systematic comparison of two or more phenomena samt hypotetiskt-deduktiv metod som beskrivits ovan. Strauss och Corbin föreslår

[59] Strauss, Anselm och Juliet Corbin, *Basic qualitative research: Techniques and procedures for developing grounded theory*.
[60] Merriam, B Sharan, *Fallstudien som forskningsmetod.*
[61] Strauss, Anselm och Juliet Corbin, *Basic qualitative research: Techniques and procedures for developing grounded theory*.
[62] Ibid.
[63] Merriam, B Sharan, *Fallstudien som forskningsmetod*.
[64] Strauss, Anselm och Juliet Corbin, *Basic qualitative research: Techniques and procedures for developing grounded theory*.

jämförelsetekniken Systematic comparison of two or more phenomena då en jämförelse mellan en studerad händelse och till exempel tidigare erfarenheter som finns i litteratur är intressant.[65]

Då ett problem uppstått under studiens experiment har detta blivit en ingående observerad premiss i en hypotetiskt-deduktiv struktur. Hypoteser kring problemets giltighet har sedan hämtats från lämplig litteratur och sakenliga forum. Med hjälp av de premisser och hypoteser som ingått har problemets giltighet antingen bestyrkts eller avstyrkts. Då problemet bestyrkts har detta jämförts med de fall som tidigare inträffat. De egenskaper som varit lika och de som varit olika har därefter antecknats som hypoteser och kontrollerats mot ytterligare litteratur och forum. Problemen och deras lösningar gicks sedan igenom ännu en gång och verifierades eller falsifierades enligt hypotetisk-deduktiv-metod. Denna jämförelse minskar risken för att viktiga egenskaper berörande problemet förbises.[66] Då ett problem uppstått och belägg inte kunnat hittas i litteraturen trots åtskilliga försök har problemet undersökts återigen och frågor ställts i lämpliga forum. Om problemet verkligen varit ett problem har det, trots att ingen litteratur kring ämnet hittats, kategoriserats och behandlats som en observerad premiss och som de bestyrkta problemen.

Kontinuerligt kontrollerades att kategorierna och kategoriseringen fortfarande var aktuell. För att få en övergripande bild av de kategorier som hittats, bekräftades och skrevs alla de egenskaper de olika kategorierna hade ner.

Litteraturstudie

Vid en studie som denna är det viktigt att ta hänsyn till tidigare arbeten som utförts inom forskningsområdet. Om detta inte görs finns risk för att ett trivialt problem studeras, en kopia av en tidigare undersökning produceras eller att andras misstag upprepas. Målet för forskningen, att bidra till att vidga kunskapsbasen inom det aktuella området, kommer då kanske inte att bli uppfyllt.[67]

Nedan följer en lista på de databaser och sökmotorer som använts mest frekvent under denna studie.

DATABAS/SÖKMOTOR	BESKRIVNING	WWW-ADRESS
Altavista	Internetsökmotor.	http://www.altavista.com
Amazon	Sökmotor till engelsk litteratur.	http://www.amazon.com
Association for Computing Machinery (ACM)	Portal för datorlitteratur och vetenskapliga artiklar inom samma område.	http://www.acm.org/
Bitpipe	Portal för white papers, fallstudier m.m inom informationstekonologi.	http://www.bitpipe.com/
Chans	Chalmers biblioteks sökmotor.	http://www.lib.chalmers.se/

[65] Strauss, Anselm och Juliet Corbin, *Basic qualitative research: Techniques and procedures for developing grounded theory*.
[66] Ibid.
[67] Merriam, B Sharan, *Fallstudien som forskningsmetod*.

CiteSeer	Databas som fokuserar på vetenskapliga artiklar inom datavetenskap och informationstekonologi.	http://citeseer.ist.psu.edu/
CrossRef Search	Internetsökmotor för vetenskaplig litteratur.	http://www.google.com/cobrand?restrict=crossref&filter=0&cof=AWPID:bbd6d01e9a530922
Electronic Journals Service	Databas med forskningsartiklar.	http://ejournals.ebsco.com/
Emerald Insight	Databas med vetenskaplig litteratur bland annat inom teknik.	http://ceres.emeraldinsight.com/
Google	Internetsökmotor.	http://www.google.com och http://www.google.se
Google scholar	Internetsökmotor för vetenskaplig litteratur.	http://scholar.google.com/
Gunda	Göteborgs universitetsbiblioteks sökmotor och databaser.	http://webbgunda.ub.gu.se:8000/cgi-bin/chameleon
Libris	Sökmotor innehållande det mesta inom svensk litteratur.	http://www.libris.kb.se
Science Direct	Databas med vetenskaplig litteratur bland annat inom teknik.	http://www.sciencedirect.com/

Tabell 3 Översikt över de databaser och sökmotorer som använts

Litteraturstudien resulterade i ett för uppsatsen teoretiskt ramverk. Fundamentet för experimentet grundar sig både på det faktum att målet för uppsatsen är att uppmärksamma problem som kan uppstå under en migrering, samt det faktum att målet för migreringen är en applikation som är möjlig att exekvera på ett flertal plattformar med hjälp av *en* lösning, det vill säga, applikationen ska vara plattformsoberoende. Det teoretiska fundamentet presenteras här nedan.

Teori

Avsnittet omfattar beskrivning av, och riktlinjer för, migreringen som process. För att belysa programmeringsaspekter beskrivs programspråket Java, dess arkitektur och plattformsoberoende egenskaper. Avsnittet inkluderar även ett antal faktorer att ta hänsyn till vid migrering, dels programspråkens olika egenskaper, dels skillnader mellan målplattformar. Avsnittet innehåller även en översikt över några av de format som kan utnyttjas för att bevara en applikations plattformsoberoende egenskaper.

Migrering

Migrering av kod är en process där man behandlar och förvandlar en befintlig applikation på sådant sätt att resultatet utgör en mjukvara som kan exekveras tillfredsställande på en ny plattform. Här fokuseras i första hand på att möjligheterna att kunna utnyttja en mjukvaras funktionalitet genom att möjliggöra exekvering under ett flertal operativsystem, såsom Windows, Mac OS eller Linux. Det man vill uppnå är att skriva program en gång, som har möjligheten att exekveras på ett flertal olika datorer och på ett flertal olika operativsystem utan att programmet behöva förändras i relation till plattformen.[68] Det finns dock ytterligare fördelar, och nackdelar, med migrering i jämförelse med nyskapande.

Fördelar och nackdelar

Vid en migreringsprocess har man fördelen av att stora delar av systemutvecklingsarbetet redan är utfört. Man har tillgång till specifikationer, applikationens arkitektur och design är redan utvecklade och man har möjlighet att både studera den befintliga applikationen som då utgör en fullständigt funktionell prototyp, samt dess källkod. Processens grad av komplexitet avgörs av egenskaperna hos den ursprungliga mjukvaran och dess plattform samt den plattform som är målet för förändringen. Dessa egenskaper påverkar huruvida migreringen blir en enkel förflyttning, eventuellt bestående av en kompilering av källkoden på den nya plattformen, eller en komplex process som kräver ny design och omskrivning och anpassning av stora delar av källkoden till den nya plattformen.[69]

En del av de tidsbesparande faktorerna som nämns ovan kan dock gå förlorade i form av att man istället får ägna tid åt förstudier i form av att förstå den ursprungliga källkoden och identifiera delsystem i applikationen. De specifikationer man har tillgång till är inte sällan skrivna för att passa det ursprungliga ramverket eller den plattform applikationen i utgångsläget utvecklats för. Dessutom finns alltid en risk för att arkitektur och design är bristfälligt utvecklade.[70] Man menar till och med att, om det handlar om en fullständig migrering av en applikation, är nyskapande att föredra framför migrering. Och i de fall delar av en applikation ska migreras, kan man räkna med att stöta på problem under migreringens gång, men om migreringens syfte är portabilitet, kan det ändå vara värt besväret.[71]

Migreringsformer

Det finns flera former av migrering. Migrering kan vara en möjlighet förändra befintliga applikationer för att få möjlighet att utnyttja bland annat andra[72]

- Operativsystem, eller olika versioner av sådana

[68] Crawford, Michael D, *Writing Cross-Platform Software – Getting Started*, 2002
[69] Mindfiresolutions, *Porting: A development primer*.
[70] Ibid.
[71] Kalev, Danny, *Porting a C++ Application to Java,* 1998
[72] Mindfiresolutions, *Porting: A development primer*.

- Databaser
- Språk
- Ramverk och bibliotek
- Teknologier
- Utvecklingsverktyg
- Webbplattformar

Migrering innefattar ofta ett flertal av ovanstående typer. Det uppsatsen i första hand fokuserar på här, är migrering för att uppnå plattformsoberoende, det vill säga att göra det möjligt att tillfredsställande kunna exekvera en applikation på ett flertal operativsystem. Flera av delarna kommer dock att vara avgörande i processen vilket påvisas i kommande avsnitt.

Migreringsprocessen

Det är viktigt att förstå och klargöra målen för migreringen.[73,74] Först bör man klargöra vilken typ av migrering som är att vänta. Olika nivåer för migreringen kan vara som följer:

- Att få en enkel applikation som fungerar tillfredsställande på den nya plattformen, vilken senare byggs på till en fullt funktionell version.
- Att göra en fullständig migrering där applikationen i sin helhet migreras till den nya plattformen.
- Att migrera den existerande applikationen och bygga ut den med ytterligare funktioner för målplattformen. Det ses ofta som en fördel vid migrering att man omvärderar kravspecifikationer och design, vilket möjligt leder till förbättringar även i den ursprungliga applikationen.[75]

Oavsett vilket av ovanstående man väljer är det viktigt att man sedan följer planen för migreringen. Att sätta upp kriterier för att kunna mäta framstegen ger en bra bas för de val som måste göras under migreringen, och gör det möjligt att genom en objektiv bedömning avgöra när migreringen kan anses slutförd.[76]

Härnäst bör man utreda hur den ursprungliga applikationen kommer att bete sig på den nya plattformen. Sällan är det möjligt att göra en direkt översättning av koden. Troligare är att vissa funktioner saknar motsvarighet eller till och med relevans på den nya plattformen. Det kan också vara så att målplattformen har egna restriktioner eller fördelar som kan vara värda att ta hänsyn till. Om det därför krävs ändringar i de ursprungliga specifikationerna, ska dessa göras.[77]

För att kunna utföra en migrering krävs en förståelse för den ursprungliga applikationen. Här kan man ta hjälp av dokumentation, reverse engineering eller enkla diagram som ger en bild av applikationens struktur, omfattning, beroenden och funktionalitet.[78] En bra insikt i den ursprungliga applikationens funktionalitet gör det också lättare att uppfatta faktorer så som eventuella prestandaförluster som kan uppstå på den nya plattformen, vilket kan tvinga till ytterligare arbete när koden är migrerad.[79] Om applikationen har en god design är den troligt

[73] Mindfiresolutions, *Porting: A development primer*.
[74] Cravotta, Robert, *Fear and loathing of porting embedded software.*
[75] Mindfiresolutions, *Porting: A development primer*.
[76] Cravotta, Robert, *Fear and loathing of porting embedded software.*
[77] Mindfiresolutions, *Porting: A development primer*.
[78] Ibid.
[79] Cravotta, Robert, *Fear and loathing of porting embedded software.*

uppdelad i ett antal lager, med gränssnitt däremellan. Att behålla denna struktur reducerar den mängd kod som kommer att behöva förändras på den nya plattformen.[80]

Valet av verktyg vid migreringen kan vara ett avvägande på grund av att verktyg som finns tillgängliga både för den ursprungliga plattformen och för målplattformen är att föredra men ändå inte alltid är det bästa alternativet. Om verktyget av någon anledning inte är tillräckligt kraftfullt och det för målplattformen finns bättre verktyg att tillgå, kan det vara aktuellt att lägga ner extra tid på att introducera det nya verktyget för tidsbesparing i längden.[81]

Migrering handlar inte bara om att få applikationen att fungera tillfredsställande på den nya plattformen utan även om att revidera den tidigare utgåvan. Under arbetets gång kan buggar och felaktigheter i både den gamla och den nya versionen behöva tas om hand. Det här kan exempelvis bero på att man använder sig av en nyare utgåva av kompilator, som inte godtar koden i dess ursprungliga skick.[82] I det här stadiet kan det också vara bra att rätta till fel man är medveten om i den ursprungliga källkoden, både för den tidigare och senare utgåvan, om möjligt.[83]

Det rekommenderas att man, när man migrerar till portabel kod, strävar efter att uppnå samma bas av källkod på båda utgåvorna, för att underlätta framtida uppdateringar och underhåll. Detta kan åstadkommas genom att man förändrar även den ursprungliga källkoden i det fall man tvingas ändra koden för den nya plattformen.[84]

Detta gäller även i fall då den ursprungliga applikationen är beroende av hjälpfiler och dessa är plattformsspecifika. Då bör dessa förändras till att vara av ett format som passar den nya plattformen eller är plattformsoberoende. Allmänna rekommendationer vid migrering är att om man väljer att översätta dessa filer till ett annat format, bör man även överväga att förändra formatet för hjälpfilerna i den ursprungliga applikationen för att få samma bas för båda versionerna.[85]

Ytterligare en del av applikationen, användargränssnittet, behöver ses över. Även om det användargränssnittet är väl designat och fullt funktionellt kan det vara stor skillnad på det grafiska utförandet mellan olika plattformar. Gränssnittet ska förändras för att passa den nya plattformen och samtidigt behålla full funktionalitet.[86] Att bygga olika gränssnitt för varje aktuell plattform är ett alternativ som skulle göra att varje gränssnitt skulle kunna anpassas perfekt till plattformen. Den lösningen skulle dock kräva mycket arbete och resultatet skulle inte vara plattformsoberoende. Att använda ett portabelt verktyg för GUI:s vore en bättre lösning. Dessa finns i två utföranden, wrapper och emulated (pure). Wrappers handhar komponenter för varje specifik plattform och har samma funktionalitet, men saknar egenskapen av flexibilitet. Den egenskapen har däremot de som kallas emulated, som tar hand om plattformarnas anrop för att rita, och använder dessa för att skapa komponenter. [87]

När den förberedande fasen är avklarad är det dags att välja migreringsstrategi.

---

[80] Mindfiresolutions, *Porting: A development primer*.
[81] Ibid.
[82] Cravotta, Robert, *Fear and loathing of porting embedded software.*
[83] Mindfiresolutions, *Porting: A development primer*.
[84] Ibid.
[85] Ibid.
[86] Ibid.
[87] Knight, David, Thurley, James och Wittams, Rob, *Platform Independence – Final report*, 2001.

Migreringsstrategier

Det finns ett flertal migreringsstrategier, inte sällan modifierade för varje specifik migrering. Det finns dock vissa riktlinjer som är applicerbara på flertalet strategier.

- Att ha en exekverbar källkod klar för den nya plattformen i ett så tidigt skede som möjligt är att föredra.[88]
- Förbered målplattformen genom att göra det tunga jobbet först; hantera resurser, hjälpfiler, eventuella wrappers för kod och datatyper och liknande. [89]
- Välj kodstrategi för att underlätta arbetet. [90]
- Planera tidigt för hur arkitekturella skillnader ska hanteras.[91]
- Se till att möjligheten finns att börja om. Om något inte går planenligt bör man kunna återgå till en korrekt punkt, och fortsätta arbetet från den punkten.[92]
- Arbeta för att få *en* källkodsbas. Om portningsprojektets struktur utvecklar sig olikt från grundstrukturen, kommer resultatet bli ett tidskrävande arbete när man vid underhåll och uppdateringar har två versioner att förändra.[93]

Den vanligaste strategin vid portning är att helt enkelt kompilera om koden med hjälp av en kompilator på målplattformen, och därefter översätta de delar av koden som inte är portabla.[94]

Valet av migreringsstrategi är att väga fördelar och nackdelar med olika förfaringssätt mot varandra i relation till de förändringar av den ursprungliga designen och källkoden som förutsätts göras. De fyra första strategierna som diskuteras nedan är förhållningssätt till i vilken omfattning den ursprungliga källkoden förändras under tiden migreringen pågår. De därefter följande beskriver strategier för omfattningen av den ursprungliga källkod som bevaras respektive översätts vid migreringen, samt strategi för användande av verktyg för en automatiserad process.

Freeze and port

Här väljer man att helt enkelt göra en kopia av källkoden och under hela migreringen arbeta mot denna, och inte under tiden göra uppdateringar i den ursprungliga källkoden. Här har man möjlighet att fokusera helt på migreringen, vilket är en fördel.[95]

Branch and port

Liknar ovanstående strategi, men med undantaget att man arbetet delas upp i två där ett team gör kontinuerliga uppdateringar i den ursprungliga källkoden under tiden ett annat team utför migreringen. Den här strategin innebär att man förr eller senare måste utföra uppdateringarna som skett även på den migrerade koden, vilket beroende på mängden arbete som utförts kan bli en enklare, eller svårare uppgift.[96]

---

[88] Mindfiresolutions, *Porting: A development primer*.
[89] Ibid.
[90] Ibid.
[91] Ibid.
[92] Kelley, Allan, *Where to begin: Porting*.
[93] Ibid.
[94] Cravotta, Robert, *Fear and loathing of porting embedded software*.
[95] Kelley, Allan, *Where to begin: Porting*.
[96] Ibid.

Creeping port

Förhoppningsvis består den ursprungliga källkoden av ett antal lager. Här utnyttjar man möjligheten att utföra freeze and port på ett lager i taget, där man arbetar från botten och uppåt vilket göra att man kan testa lager för lager efterhand som migreringen utförs.[97]

Proto-type port

Den här strategin inriktar sig på att före migreringen göra en quick and dirty-migrering vars syfte är att på kort tid uppnå en exekverbar prototyp på den nya plattformen. Detta görs för att uppfatta omfattningen av, och egenskaper hos, de problem som kommer att behöva hanteras vid migreringen. Det här kan vara ett bra tillvägagångssätt om man inte tidigare utfört någon migrering och behöver få en överblick över arbetet man står inför.[98]

Integration of native binary code

Att migrera koden så att den integreras i Java med hjälp av nativekod är enkelt på så sätt att koden inte behöver förändras nämnvärt. Men det är nödvändigt att implementera gränssnittsklasser som kan hantera kommunikationen mellan C++ och Java-koden . Dessa gränssnitt kan skapas mer eller mindre automatiskt, men då de under exekvering ständigt får hantera datatypomvandlingar, kommer det ha inverkan på prestanda, vilken då försämras. Denna prestandaförsämring bör då vägas mot prestandafördelarna med att integrera C++-kod i Java. Det finns ytterligare nackdelar med denna typ av migreringsstrategi. Lösningen kommer att förlora sina plattformsoberoende egenskaper och kommer endast att kunna exekveras under utvecklingsplattformen. Eftersom C++-koden inte exekveras under Javas virtuella maskin är den inte skyddad mot säkerhetshot och felaktigheter i samma grad som den varit om den funnits inom Javas virtuella maskin.[99]

C to byte code compilation

Javas virtuella maskin var designad med Java som basprogramspråk, men ger även möjlighet att utnyttja andra språk. Trots att det här kan vara en till synes bra strategi har den implementerats på sätt som gjort integreringen av C++ kod svårhanterlig och har kompromissat med Javas typsäkerhet. För att hantera minneshanteringen i C++ korrekt under Javas virtuella maskin och komma runt den strikta typkontrollen i Java, implementeras en omfattande `array` som lagrar typerna i C++. Som diskuterats i föregående strategi kan felaktiga program orsaka att denna `array` blir korrupt och orsakar oväntade fel och svårigheter med debuggning. Ett annat problem med detta förfarande är att man behöver implementera speciella gränssnittsklasser för att hantera kommunikationen mellan C++-koden och Javakoden. Stödrutiner krävs för att skapa Java-objekt från rådata i `arrayen`. Som tidigare diskuterats kommer detta resultera i prestandaförsämringar.[100]

Re-implementation

Ett annat sätt att utföra migreringen på, är att använda den ursprungliga designen och dokumentationen för att skriva om applikationen i målprogramspråket. På grund av att en applikation förmodligen utvecklas över tiden, och vanligtvis under tidspress, är det inte omöjligt att de ursprungliga dokumenten inte uppdaterats i takt med källkodens utveckling. Om så är fallet får dessa dokument återskapas med hjälp av reverse engineering. När man återskapat designen är det möjligt att implementera denna på det nya programspråket, förmodligen efter uppdateringar beroende på önskade förbättringar eller anpassning till

[97] Kelley, Allan, *Where to begin: Porting*.
[98] Ibid.
[99] Martin, Johannes, *Ephedra A C to Java Migration Environment,* 1996.
[100] Ibid.

egenskaper i det nya programspråket. Graden av abstraktion utgörs av hur mycket man behöver göra om den ursprungliga designen. Det här är ofta förfaringssättet i de fall den ursprungliga applikationen har prestanda- eller underhållningsproblem, eller när man förväntar sig stora förändringar i kravspecifikationen. [101]

Source code transliteration

Om designen är tänkt att behållas intakt, eller om ytterst få förändringar förutses, finns möjlighet att utföra en mer eller mindre automatiserad process för att överföra koden mellan programspråk, och förändra den ursprungliga strukturen så lite som möjligt.[102] Detta görs med hjälp av verktyg som överför kod mellan olika programspråk med så få förändringar som möjligt.

Migreringsverktyg

Det finns få verktyg som med automatik migrerar C++-kod till Java-kod. Ett fungerande dito efterfrågas på många forum. C2J++ är ett verktyg med vilket man kan översätta C++-klasser till Java-klasser. Fördelen med detta verktyg är att alla de ändringar som skett noteras av verktyget, vartefter man kan kontroller syntaxkorrektheten hos dessa. Verktyget förutsätter att variabler och metoder samtliga ligger inom klasser, varför någon modifiering kan vara nödvändig innan verktyget utnyttjas till översättning. Trots språkens likheter finns också en hel del olikheter. Det är när verktyget stöter på dessa olikheter det får svårigheter att skapa en felfri syntaxmotsvarighet i Java. Verktyget kan inte skilja på om en stjärna motsvarar ett multiplikationstecken eller om den används i samband med en pekare. Den semantiska skillnaden mellan språken vad gäller tilldelning och användande av parametrar ignoreras, vilket gör att funktionaliteten inte blir densamma efter en översättning. Detta leder till att en automatisk översättning av en större mängd kod troligen leder till ett omfattande manuellt arbete för att revidera de utförda ändringarna.[103]

Migrering från C++ till Java

Programspråken C++ och Java ser vid en första anblick inte ut att skilja sig så mycket från varandra. Att Java har utvecklats med C++ som utgångspunkt är naturligtvis orsaken till detta.[104] C++ och Java har gemensamma betydelser hos bland annat primitiver (`int, short, long` osv), konstruktioner (`if, else, for` osv), operatorer (`+, -, !, %` osv) och kommentarer (`/*…*/, //`)[105]. Så långt allt väl, men det finns också en hel del skillnader i språkens syntax, där namn i språken har olika betydelser, eller där funktioner i ett språk helt enkelt saknar motsvarighet i det andra. Vissa funktioner i C++, såsom pekare och överlagrade operatorer, har avsiktligt utelämnats i Java för att göra språket mindre, lättare att lära och använda. Java är ett koncist språk som ger möjlighet att utnyttja fördelarna med objektorientering och har ett antal användbara standardbibliotek.[106] Här diskuteras förekomsten av skillnader i språken som man bör ta hänsyn till. De flesta skillnaderna att uppmärksamma vid portningen är troligtvis syntaktiska, och kompilatorn kommer att upptäcka felen. Det finns dock utrymme för att göra misstag där samma kod i båda språken kan användas, men kan ha förödande konsekvenser.[107]

---

[101] Martin, Johannes, *Ephedra A C to Java Migration Environment,* 1996.
[102] Ibid.
[103] Ibid.
[104] Eckel, Bruce, *Comparing C++ and Java*, 2000
[105] Davis, Mark, *Porting C++ to Java,* 1997
[106] Prashant, Jain och Schmidt, Douglas C, *Experiences Converting a C++ Communication Software Framework to Java*, 1997
[107] Davis, Mark, *Porting C++ to Java.*

Filstruktur

Innan portningen påbörjas bör man se till att lägga grunden till mappstrukturen som ska användas och ha en mall för hur paket och filer ska namnges. C++-källkod består ofta av två filer, en med filändelsen h, en så kallad inkluderingsfil, och en med ändelsen cpp, en så kallad definitionsfil.[108] Den exekverbara koden finns, tillsammans med en rad som inkluderar h filen och dess typdeklarationer, i filen med ändelsen cpp. I C++ finns det möjlighet för ett flertal cpp-filer att inkludera en och samma h-fil.[109] Denna uppdelning av exekverbar kod och definitioner gör koden tydlig att läsa, h-filen ger en överblick av vilka funktioner som finns tillgängliga utan att behöva avslöja detaljer om hur de egentligen utförs. I Java finns motsvarande funktionalitet genom ett verktyg, javap, som ger möjlighet att få fram information om klassens deklaration. Därför är det inte nödvändigt att använda sig av två skilda filer i Java.[110] Därför sammanfogas de båda C++-filerna till en java-fil vid portningen.

Preprocessor

Vid kompilering av C++ finns ett stadium där preprocessorn utför en funktion som används främst för att undvika att få dubbla uppsättningar av samma kod. Preprocessorn kontrollerar bland annat:

- Filer som inkluderats med hjälp av `#include` (h-filer).
- Macron – konstanter eller små funktioner som ska inkluderas i källkoden.
- Fördefinierade konstanter som deklarerats först i källkodsfilerna med hjälp av `#define`. Förekomster av dessa variabler i källkoden byts mot det definierade värdet.
- Fördefinierade typer som deklarerats med hjälp av `#typedef`.

Motsvarande inkludering av filer och paket i Java sker med uttrycket `import`.[111] Macron som definierats i C++ kan vara svåra att använda på liknande sätt i Java.[112] Beroende av dess form får dessa istället inkluderas som konstanter eller funktioner i klasser. Konstanter i Java diskuteras ytterligare i kommande avsnitt.

Tillgänglighet

När källkodsfilerna sammanfogats till en ska nyckelorden `public`, `private` eller `protected` läggas till först i definitionen av variabler och metoder beroende på deras tillgänglighet. I C++ deklareras i klassen samtliga privata variabler under en rubrik `private`, och motsvarande med publika variabler under `public`. I Java deklareras tillgängligheten för varje variabel.[113]

Vänner

I C++ finns möjlighet att definiera `friend`-klasser, vilket innebär att en klass har tillgång till variabler och metoder i en annan, oavsett vilken tillgänglighet medlemsvariablerna i denna klass har. Samma tanke finns i Java, men fungerar lite annorlunda. I Java kan klasser tillåtas access av variabler och metoder i en annan klass inom samma paket genom att man deklarerar default-tillgänglighet för dessa variabler och metoder. Default-tillgänglighet når man i Java

---

[108] Skansholm, Jan, *C++Direkt*, (Lund, Studentlitteratur, 2000).
[109] Cooperman, Gene, *Converting from Java to C++*, 2003.
[110] Vanderburg, Glenn L, et al, *Tricks of the Java Programming Gurus*, 1996.
[111] Litvin, Maria and Litvin, Gary, *Java Methods,* 2001.
[112] Martin, Johannes, *Ephedra A C to Java Migration Environment.*
[113] Vanderburg, Glenn L, et al, *Tricks of the Java Programming Gurus*.

genom att helt enkelt utesluta tillgänglighetsdeklaration i form av `public`, `private` eller `protected`.[114]

Arv

Arv i C++ markeras med **:** (kolon) mellan klass och superklass medan det i Java betecknas med `extends`. I C++ kan en klass ärva egenskaper från ett flertal klasser, vilket inte är tillåtet i Java. Javas motsvarighet till funktionaliteten multipelt arv är möjligheten att använda sig av `interface` – gränssnitt. Arv i C++, som tidigare beskrivits, skrivs med hjälp av kolon efter klassnamnet och därefter en rad av klasser som man ärver från. I Java kan endast en klass ärvas med hjälp av `extends`. Därefter kan man implementera ett obegränsat antal `interface` med hjälp av `implements`. `Interface` motsvaras av abstrakta klasser, därför, om alla utom en av klasserna i den ursprungliga källkoden är abstrakta, kan man utnyttja `interface` för att uppnå motsvarande funktionalitet i form av multipelt arv. Dessa gränssnitt innehåller alltså bara en uppsättning definierade metoder som implementeras av en eller flera klasser. I gränssnitt deklareras bara metoder och konstanter. Här kan man inte deklarera variabler.[115]

Typer

C++ har tre fördefinierade flyttalstyper; `float`, `double` och `long double`. C++ har också åtta olika fördefinierade heltalstyper, vilka är uppdelade i två grupper. Den ena gruppen består av `signed char`, `short int`, `int`, och `long int`. Den andra innehåller motsvarande typer `unsigned`. Typer med tecken, `signed`, tillåter både positiva och negativa tal, medan typer utan tecken, `unsigned`, endast tillåter lagring av positiva tal.[116] Javas primitiver saknar varianter av `signed` och `unsigned`, `char` är alltid `unsigned`, de andra `signed`. Javas `char` är större än C++:s motsvarighet varför det inte möter några hinder att ta bort ordet `signed`. De typer som är `unsigned` ska förändras till en större typ om det är så att deras storlek är avgörande.[117]

C++ skiljer inte på små tal eller bokstäver vid deklaration av en `char`. Oftast behöver de inte ändras eftersom det vanligtvis är det just en bokstav som avses. I annat fall bör man avgöra om exempelvis typen `byte` i Java bättre motsvarar det värde som avses i C++.[118]

C++ typ av `boolean` fungerar något annorlunda jämfört med motsvarande typ i Java. I C++ finns möjlighet att utnyttja typen `integer` där 0 representeras av `false`, och resterande värden är `true`. Java kan inte ta emot motsvarande värden och avgöra huruvida de är sanna eller falska. I Java används enbart de reserverade orden `true` och `false`, som man utnyttjar för samma funktionalitet.[119]

C++ använder sig av strängar i form av både standardklassen `string`, samt som arrayer av bokstäver, `char`. Java använder sig, liksom C++, `String` och `char`-arrayer.[120] Skillnaden mellan C++:s och Javas `String` är att Java har inbyggt stöd för `String` och det ger ett antal fördelar. `String` i Java skapas och ges access på samma sätt för alla strängar och

[114] Vanderburg, Glenn L, et al, *Tricks of the Java Programming Gurus*.
[115] Sun Microsystem, *Java Runtime Environment.*
[116] Skansholm, Jan, *C++Direkt*.
[117] Davis, Mark, *Porting C++ to Java.*
[118] Ibid.
[119] Vanderburg, Glenn L, et al, *Tricks of the Java Programming Gurus*.
[120] Gosling, J. et al, *The Java Language Specification*, 2005.

eftersom `String` är en del av Javas språk har de förutsägbar funktionalitet. I de fall `char`-arrayer används, bör de ändras efterhand som de dyker upp i koden och konverteras till `String` i Java med motsvarande funktionalitet.[121]

I C++ kan typer och tillåtna värden variera beroende av vilket system som används. Värden nedan är de allmänt accepterade och de som används av de flesta kompilatorerna.[122]

	JAVA		C++	
TYP	**BYTE**	**RÄCKVIDD**	**BYTE**	**RÄCKVIDD**
boolean	1	true eller false	1	true or false
byte	1	signed: -128 till 127	1	signed: -128 till 127 unsigned: 0 till 255
char	2	unsigned: 0 till 65535	1	signed: -128 till 127 unsigned: 0 till 255
double	8	Ungefär 1.7E +/- 308 (15 siffrors precision)	8	1.7E +/- 308 (15 siffrors precision)
float	4	Ungefär 3.4E +/- 38 (7 siffrors precision)	4	3.4E +/- 38 (7 siffrors precision)
int	4	signed: -2147483648 till 2147483647	*	Varierar i storlek beroende på operativsystemet. I Windows är den 4 byte och i MS DOS 2 byte
long	8	signed: -9223372036854775808 till 9223372036854775807	4	signed: -2147483648 till 2147483647 unsigned: 0 till 4294967295
short	2	signed: -32768 till 32767	2	signed: -32768 till 32767 unsigned: 0 to 65535

Tabell 4 Typer i Java och C++[123,124]

Typomvandling

I C++ är det tillåtet att ge exempelvis en `int` värdet av en `float` och därigenom riskera att förlora precision. I Java är det inte tillåtet, kompilatorn kommer att säga ifrån. Om man vill göra en typomvandling i Java måste den göras explicit, det vill säga man måste tydliggöra vilken typ man vill omvandla till och försäkra kompilatorn om att man uppmärksammat eventuella precisionsförluster.[125]

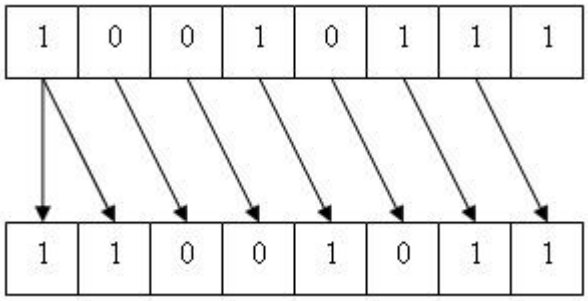
[121] Vanderburg, Glenn L, et al, *Tricks of the Java Programming Gurus*.
[122] Soulié, Juan, *The cplusplus.com language tutorial,*2003
[123] Skansholm, Jan, *Java direkt*. 2:a upplagan, Lund: Studentlitteratur, 1999.
[124] Soulié, Juan, *The cplusplus.com language tutorial.*
[125] Vanderburg, Glenn L, et al, *Tricks of the Java Programming Gurus*.

Fält

`Array` i Java innehåller riktiga objekt, inte bara pekare. C++ pekarfunktioner kan därför inte användas, vilket innebär att man får använda sig av en räknare när man itererar igenom arrayer. I Java finns funktion för att hämta arrayens storlek, därför finns det möjlighet att, istället för att hårdkoda storleken, utnyttja denna funktion, vilket rekommenderas. Arrayer initieras med 0 för numeriska primitiver, `false` för boolean och `null` för objekt. Värden i arrayer skapas dock inte automatiskt, de måste bli initierade.[126]

Shift-operatorer

För att läsaren ska få en bättre förståelse för vad shift-operatorer innebär följer en beskrivning. Shift-operatorer är en typ av bitoperatorer med vilka man kan arbeta direkt med bitarna i till exempel en variabel. Det finns olika typer av shift-operationer men för denna studie räcker det med en genomgång av aritmetiska shift-operatorerna för höger och vänster shift. Shift-operatorn för höger-shift är: >> och för vänster-shift ser operatorn ut på följande sätt: <<. Operationen flyttar bitarna till vänster eller höger beroende på vilken shift-operator som används. Bilden och texten som följer förklarar på ett tydligt sätt hur en aritmetisk höger-shift fungerar:[127]

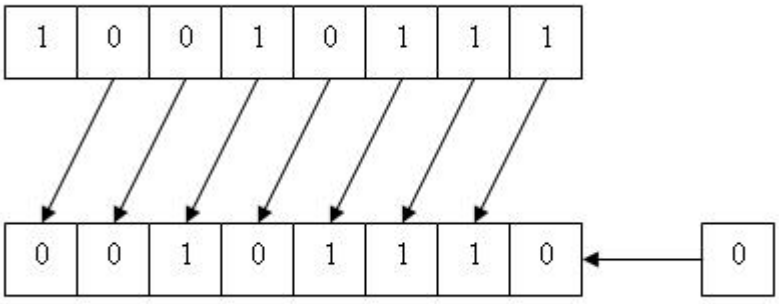


Figur 3 Aritmetisk höger-shift

Här arbetar höger-shift-operatorn på en `byte`, den översta byten visar den icke-shiftade byten och den nedre byten visar hur byten ser ut efter shift-operationen. Biten längst till vänster kopieras in längst till vänster och näst längst till vänster, de andra bitarna förflyttas ett steg åt höger och den sista biten faller utanför.[128]

Vänster-shift-operatorn fungerar, som kan ses på bilden nedan, lite annorlunda. Den bit som är längst till vänster faller utanför och längst till höger kopieras en nolla in.[129]

[126] Davis, Mark, *Porting C++ to Java.*
[127] Answers.com, *Bitwise operations.*
[128] Ibid.
[129] Ibid.



Figur 4 Aritmetisk vänster-shift

Vid användning av shift-operatorer i C++ och Java kan användaren specificera hur många steg bitarna ska shiftas.[130,131] I Java finns tre shift-operatorer:[132]

OPERATOR	BESKRIVNING
<<	Aritmetisk vänster-shift som fungerar som beskrivits ovan.
>>	Aritmetisk höger-shift som fungerar som beskrivits ovan.
>>>	Logisk höger-shift som kopierar in nollor i de vänstra bitarna.

Tabell 5 Javas shift-operatorer

Java har alltså utöver de aritmetiska operatorer som finns i C++ ytterligare en operator. Den tredje operatorn fungerar på samma sätt som en C++ höger-shift i en `unsigned` variabel. Då kod migreras från C++ till Java ska alla ställen i C++-koden där höger-shift (>>) sker med en `unsigned` variabel bytas ut mot Javas logiska högershift >>>.[133]

Konstanter

Ytterligare ett reserverat ord i C++, som saknas hos Java, är `const`. Det enklaste greppet att ta här är att byta ut ordet `const` mot `final`, men med försiktighet. Avsaknaden av `const` i Java kan ha stor betydelse för kodens kraftfullhet. I Java saknas möjligheten att avgöra rätten att modifiera objekt för varje objekt, denna avgränsning kan endast göras på klassnivå. Detta gör det svårt att uppnå samma robusthet i Java som hos C++. Det rekommenderas därför att man skriver ett gränssnitt för de objekt som inte är modifierbara för de metoder där det är önskvärt att försäkra sig om att objekten inte förändras.[134] För att uppnå konstanta variabler i Java bör man deklarera medlemsvariabler med hjälp av `static`, som innebär att bara en sådan datamedlem finns för hela klassen, och `final`, som innebär att variabeln är en konstant. Önskar man sedan att dessa statiska och konstanta variabler ska kunna nås utifrån klassen bör man även deklarera dessa som `public`.[135]

[130] Sun Microsystems, *The Java Tutorial.*
[131] Answers.com, *Bitwise operations.*
[132] Sun Microsystems, *The Java Tutoria.l.*
[133] Davis, Mark, *Porting C++ to Java.*
[134] Ibid.
[135] Vanderburg, Glenn L, et al, *Tricks of the Java Programming Gurus*.

Globaler

Det finns inte möjlighet att använda sig av globala data och metoder i Java på samma sätt som i C++. Motsvarigheten i Java är att deklarera dem i en klass och göra dem statiska med hjälp av `static`.[136]

Överlagrade operatorer

Operatorer kan överlagras i C++, men inte i Java. När man programmerar i Java bör inte detta ställa till något problem, men det kan finnas förekomster i C++ vid en migrering som man bör ta hänsyn till.[137]

Metoder

I C++ förekommer i samband med metoder olika reserverade ord såsom `virtual`, `inline` och `register`. Dessa hanteras på följande sätt. `final` ska skrivas in först i de metoder som saknar ordet `virtual` och alla instanser av `virtual` ska tas bort. Förekomsten av `virtual` ska kontrolleras i eventuella superklasser, eftersom denna egenskap ärvs ner till subklasserna. Förekomster av ordet `inline` (dessa är `final`–metoder) ska tas bort, samt förekomsten av ordet `register` (som enbart är en upplysning till kompilatorn).[138]

Defaultparametrar

I Java saknas möjligheten att använda sig av defaultparametrar. Om man känner det nödvändigt att bevara dem bör man istället definiera en överlagrad metod för varje parameter, som istället för en variabel returnerar det värde man önskar som default.[139]

Kommandoradsargument

Argument som skickas till programmet skiljer sig något mellan programspråken. I C++ finns två typer som tar hand om dessa argument, `argv` och `argc`. `argv` är en pekare till en `char`-array som innehåller det faktiska argumentet. `argc` innehåller värdet av antalet argument som skickats. I Java finns bara en typ som tar hand om argumenten, nämligen `args`. `args` är en `array` av `String` som innehåller alla argument. Det första argumentet som skickas i C++ är namnet på programmet, vilket inte behöver skickas i Java, eftersom programmets namn är detsamma som för klassen.[140]

Pekare

Pekare som används i C++ saknar motsvarighet i Java. Men när man gör en portning från C++ till Java bör man tänka omvänt; *alla* objekt är pekare i Java. Att tilldela ett objekt a värdet av ett objekt b tilldelar egentligen inte ett värde, utan bara en pekare från a till b. För att få ett nytt objekt och tilldela objektet det egentliga värdet måste man använda metoden `clone()`. En jämförelse av objekt a och objekt b med hjälp operatorn == blir endast en jämförelse av pekare. För att jämföra objektens verkliga värde måste man använda sig av metoden `equals()`.[141]

---

[136] Eckel, Bruce, *Thinking in Java Excerpt.*
[137] Davis, Mark, *Porting C++ to Java.*
[138] Ibid.
[139] Ibid.
[140] Vanderburg, Glenn L, et al, *Tricks of the Java Programming Gurus*.
[141] Davis, Mark, *Porting C++ to Java.*

Referenser

C++ har också referenser vars motsvarighet saknas i Java. Dessa kan användas som in- och utparametrar till metoder. Att förändra en inparameter från C++ till Java bör inte vara svårare än att ta bort ordet `const` i metoden. Utparametrar däremot, är något mer komplicerade i C++. För att uppnå samma funktionalitet i Java har man tre möjligheter.

- Returnera det värdet man önskar förändra
  Med C++ möjlighet att förändra referenser är det troligt att man använt en på grund av att return-satsen redan används av ett annat värde.
- Använd arrayer
  En lösning, om ej någon vacker sådan, är att använda sig av en array som parameter till en metod. Arrayer är alltid möjliga att förändra, man kan alltid hämta eller sätta första värdet i `arrayen`.
- Skapa nya klasser
  En möjlighet som förvisso kräver mer arbete, men kan anses vara en något bättre lösning än att använda arrayer, är att skapa nya klasser med fält som motsvarar parametrar och returvärden. Att göra dessa klasser förändringsbara gör det möjligt att använda dem som utparametrar och förändra dem.[142]

Minneshantering

I C++ deklareras `delete`-metoder och destruktorer för att hantera de objekt man önskar ta bort. Javas har en så kallad Garbage Collector, en skräpsamlare, som sköter detta, och det är därför inte nödvändigt att bevara dessa metoder och destruktorer när man migrerar från C++. Man kan använda metoden `finalize` och överlagra denna för att utföra åtgärder och ta bort ett objekt. `Finalize` anropas dock inte när objektets livstid är över, utan först när skräpsamlaren blivit tillsagd att ta hand om objektet. En rekommendation är därför att skriva koden så att minneshanteringen i sådana här fall inte är en avgörande faktor i applikationen.[143] Om man vet att man inte ska använda ett objekt längre kan man se till att objektet saknar referens, man tilldelar variabeln värdet `null`, vilket ger systemet möjlighet att återanvända det minnesutrymme variabeln reserverat. Nackdelen med Javas typ av minneshantering är att den är tidskritisk, vilket bör jämföras med den tid som annars skulle ha tagits upp av att manuellt kontrollera och hantera objekt som inte längre utnyttjas under exekveringen.[144]

Exceptions

Exceptions är så kallade exekveringsfel. Något inträffar som inte normalt sett ska inträffa.[145] Exceptions hanteras på liknande sätt i båda språken. Skillnaden är att man, som vanligt i Java, måste skapa dem med hjälp av `new`. Exceptions fångas med hjälp av `try`- och `catch`-block. `Try`-blocket innehåller satser vilka kan generera en exceptionell händelse, medan `catch`-blocken innehåller parametrar. Vilket `catch`-block som anropas är beroende av vilka parametrar som motsvarar händelsen. Den funktion i C++ som fångar alla exceptions, `catch(…)`, måste i C++ ligga sist av `catch`-blocken, eller, om händelsens art är av betydelse, använder man som parameter en referens till klassen `exception`.[146] Dessa ska i Java ersättas med basklassen för exceptions, nämligen `Exception`. I Java är man tvingad att deklarera samtliga exceptions, förutom de som genereras från `RuntimeException` eller

[142] Davis, Mark, *Porting C++ to Java.*
[143] Kalev, Danny, *Porting a C++ Application to Java.*
[144] Skansholm, Jan, *Java direkt*.
[145] Ibid.
[146] Skansholm, Jan, *C++ Direkt*.

`Error` vilka skulle kunna uppstå i, och därför få deklareras i, princip all kod om så hade varit nödvändigt.[147]

Java

En stor anledning till att programmerare använder Java beror på drömmen om fullständigt plattformsoberoende. Java strävar efter mottot ”Write once, run everywhere”, vilket innebär att skriva en uppsättning källkod, som kan exekveras på ett flertal arkitekturer eller operativsystem.[148] Detta görs möjligt med hjälp av ett verktyg, Javas virtuella maskin, som beskrivs i kommande avsnitt.

Allmänt om Java

Java är ett objektorienterat programspråk som utvecklats av Sun Microsystems. Till en början användes det främst till att skapa så kallade Applets som användes på webbsidor,[149] men idag används det av programmerare världen över för att skapa fullständiga applikationsprogram.[150,151]

Utvecklingen av programspråket Java startade som ett projekt med syftet att göra det möjligt att skapa program som var säkra, som använde minimalt med systemresurser och som var möjliga att exekvera på vilken mjuk- eller hårdvaruplattform som helst, samt kunde utvidgas dynamiskt. Arbetet startades från början med C++ men efterhand som problem uppstod med att uppnå syftet insåg man att det var nödvändigt att skapa ett nytt programspråk, vilket influerats av ett flertal språk såsom Eiffel, Smalltalk, Objective C och Cedar/Mesa.[152] James Gosling och andra utvecklare på Sun, släppte sin första version av Java 1995. Då som ett gratis utvecklingsverktyg man kunde få tillgång till via företagets hemsida. Java blev snabbt populärt i datorvärlden och sedan dess har språket utvecklats kontinuerligt och antalet användare har ökat.[153,154] Java fick enligt undersökningar från januari 2005 lämna ifrån sig förstaplaceringen för det populäraste utvecklingsspråket till C enligt Tiobe, ett företag som specialiserat sig på att kvalitetstesta mjukvaror och som en gång per månad listar och uppdaterar de populäraste programspråken.[155] Enligt Sun är Java som programspråk ett bra val på grund av att det bland annat är enkelt, portabelt, objektorienterat, robust, säkert och dynamiskt.[156]

Javaapplikationer följer inte den vanliga proceduren vid kompilering och exekvering. Program som skrivs i C eller C++ kompileras och länkas till en enda exekverbar binärfil som är knuten till en specifik hårdvaruplattform och ett specifikt operativsystem. Detta resulterar i att en exekverbar binärfil innehållande maskinkod inte kommer att fungera på något annat operativsystem än just det operativsystem som den är kompilerad för.[157] Då maskinkoden ska exekveras placeras den exekverbara filen av operativsystemet i primärminnet och exekveras.[158] En Javaapplikation däremot kompileras till ett antal binärfiler som lagras i class-

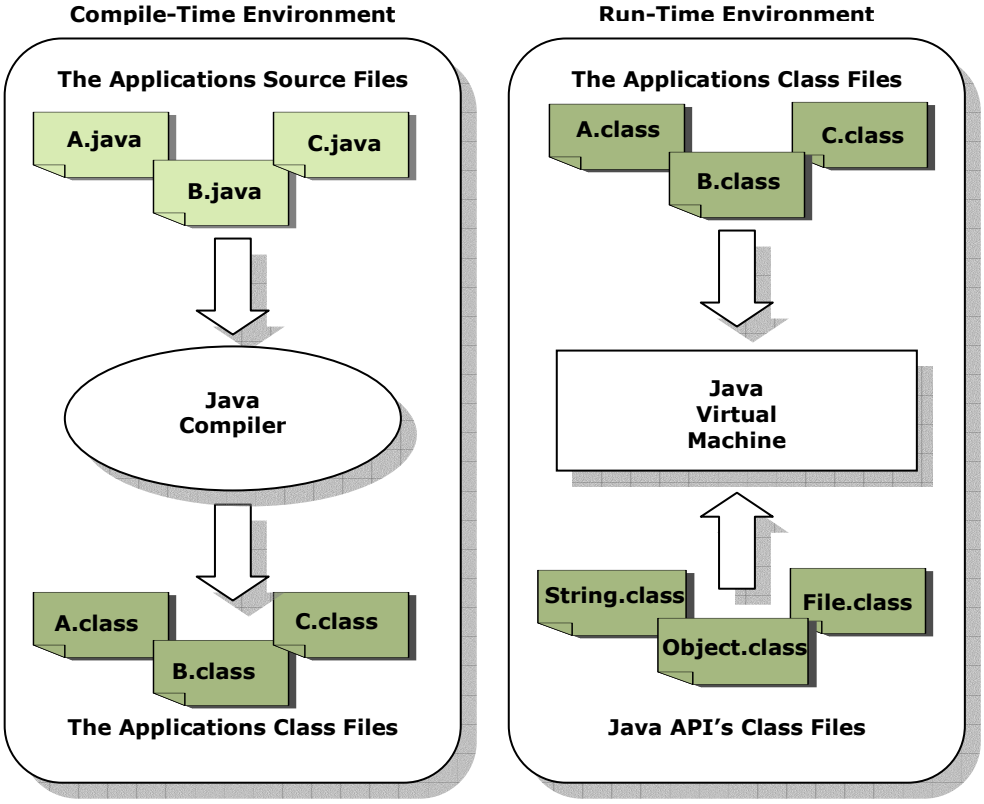
[147] Davis, Mark, *Porting C++ to Java.*
[148] Tyma, Paul, *Why are we using Java again?*, 1998.
[149] Skansholm, Jan, *Java direkt*.
[150] Ibid.
[151] Cadenhead, Rogers och Laura Lemay, *Sams Lär dig Java 2 på 3 veckor*.
[152] Gosling. James, *The Java language environment*, 1996
[153] Cadenhead, Rogers och Laura Lemay, *Sams Lär dig Java 2 på 3 veckor*.
[154] Skansholm, Jan, *Java direkt*.
[155] TIOBE, *Programming Community Index for April 2005*.
[156] Mary Campione och Kathy Walrath, *About the Java Technology*.
[157] Venners, Bill, *Inside the Java Virtual Machine*. (Addison-Wesley Professional, 1999).
[158] Skansholm, Jan, *Java direkt*.

filer. Dessa filer är kompilerade för Javaplattformen och kan därför exekveras på olika plattformar eftersom Javaplattformen kan implementeras som mjukvara på exempelvis olika operativsystem.[159,160] För att ett program ska kunna exekveras på ett operativsystem krävs att användaren installerat Javas plattform Java Runtime Environment (JRE) på sin dator.[161]

Övergripande arkitektur

Javas arkitektur kan delas upp i fyra komponenter: Javas programmeringsspråk, Javas API, Javas class-filer samt Javas virtuella maskin. Vid kompilering och körning av ett Javaprogram nyttjas alla de fyra komponenterna. Programmet skrivs i java-filer med hjälp av Javas programmeringsspråk. Kod som skrivs med hjälp av Javas programmeringsspråk anropar metoder som finns i Javas API. Kompileringen av java-filerna bildar class-filer som då programmet körs exekveras på Javas virtuella maskin och tillsammans med Javas API fungerar koden så som den bör. Javas API, som tillägnas en egen rubrik längre fram i uppsatsen, består av en stor mängd filer som kompilerats till class-filer.[162]

Javas virtuella maskin tillsammans med Javas API bildar Java Runtime Environment (JRE).[163] JRE är en typ av Java plattform där program skrivna med Javas programmeringsspråk kompileras innan de exekveras.[164] På bilden nedan visas en översikt över förloppet då java-filer kompileras till class-filer samt hur class-filerna sedan laddas in i Javas virtuella maskin tillsammans med API-class-filer.



Figur 5 Javas kompilerings- och exekveringsmiljö[165]

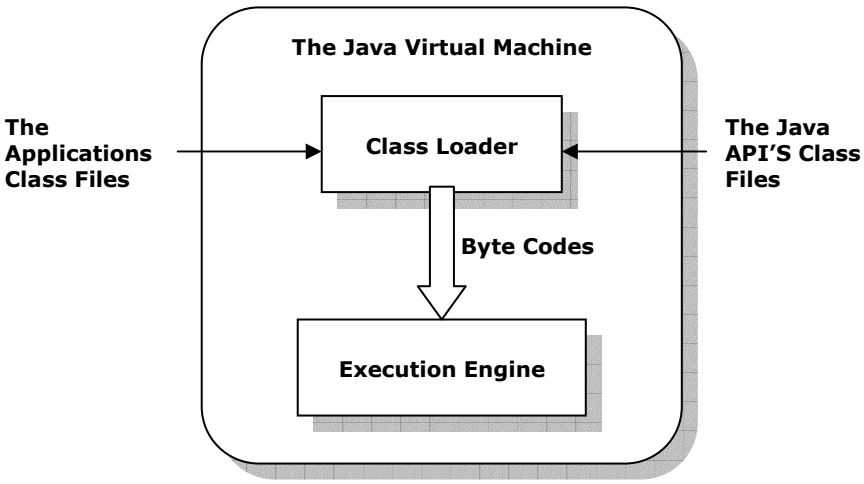
[159] Venners, Bill, *Inside the Java Virtual Machine*.
[160] Skansholm, Jan, *Java direkt*.
[161] Sun Microsystems, *Java Runtime Environment.*
[162] Venners, Bill, *Inside the Java Virtual Machine*.
[163] Sun Microsystem, *Java Runtime Environment.*
[164] Venners, Bill, *Inside the Java Virtual Machine*.
[165] Ibid.

Javas virtuella maskin

Javas virtuella maskin (JVM) är en viktig del av Javaplattformen och beskrivs med ord som Javas hörnsten, eller Javas hjärta.[166,167] JVM är den komponent som bland annat är ansvarig för att JRE är hårdvaru- och plattformsoberoende. JVM kan ses som en konstruerad dator som kan implementeras både i mjuk- och hårdvara.[168]

Som tidigare nämnts har JVM ingen vetskap om Javakoden som användaren skrivit, istället tar JVM hand om binärfiler, med filändelsen class, som innehåller JVM-instruktioner i form av bytekod.[169] JVM:s huvuduppgifter är att ladda class-filer och exekvera den bytekod som filen innehåller.

Förenklat kan förloppet förklaras med figuren nedan. Class-laddaren läser in de class-filer som tillhör applikationen och letar upp de metoddeklarationer i applikationskoden som anropar metoder i Javas API. Det är endast de API-class-filer som används av applikationen som laddas in i JVM. Applikationens class-filer och de API-class-filer som används av applikationen laddas sedan in i JVM och exekveras av en exekveringsmotor. Exekveringen kan ske på olika sätt beroende på plattform och beroende på den prestanda som programmet kräver.[170,171]



Figur 6 Förenklad bild av Javas virtuella maskin

Eftersom JVM innehåller flera class-laddare kan class-laddaren på bilden ses som ett subsystem innehållande flera class-laddare.[172]

class-filer

Då java-filerna som innehåller programkoden kompileras skapas en class-fil för varje java-fil som användaren skapat. Dessa filer är i binär form och är mjuk- och hårdvaruoberoende.[173] JVM har som tidigare nämnts ingen vetskap om Javas programmeringsspråk utan bara om det

[166] Lindholm, Tim och Frank Yellin, *The JavaTM Virtual Machine Specification* (Addison-Wesley Professional, 1999).
[167] Venners, Bill, *Inside the Java Virtual Machine*.
[168] Ibid.
[169] Lindholm, Tim och Frank Yellin, *The JavaTM Virtual Machine Specification*.
[170] Skansholm, Jan, *Java direkt*.
[171] Venners, Bill, *Inside the Java Virtual Machine*.
[172] Ibid.
[173] Lindholm, Tim och Frank Yellin, *The JavaTM Virtual Machine Specification*.

binärformat som en class-fil har. En class-fil är strukturellt uppbyggd och består av 16 delar som tillsammans innehåller all de data som JVM behöver veta om klassen eller gränssnittet.[174,175] Nedan följer en tabell som på ett överskådligt sätt förklarar en class-fils uppbyggnad:[176, 177]

NAMN	BYTE	BESKRIVNING
Magic	4	Ett tal som talar om för JVM att binärkodfilen är en giltig Javafil.
Minor version	2	Vilken underversion det var på kompilatorn som kompilerade binärkodfilen.
Major version	2	Vilken huvudversion det var på kompilatorn som kompilerade binärkodfilen.
Constant pool count	2	
Constant pool information	Variabel	Lagrar klasstrukturer, metodreferenser, namn och typ på variabler.
Access flags	2	Definierar åtkomsten för klassen (`Public`, `Protected` osv.)
This class	2	Pekare till binärkodfilens klass eller gränssnitt.
Superclass	2	Pekare till klassens superklass.
Interfaces count	2	Antalet gränssnitt som klassen implementerar.
Interfaces information	2	Pekare till klassens gränssnitt.
Fields count	2	Antalet fält som klassen innehåller.
Fields information	Variabel	Pekare till klassens fält.
Methods count	2	Antalet metoder som klassen innehåller.
Methods information	Variabel	Pekare till klassens metoder.
Attribut count	2	Antalet attribut som klassen, metoderna och fälten innehåller.
Attribut information	Variabel	Pekare till klassens attribut.

Tabell 6 class-filens uppbyggnad

API

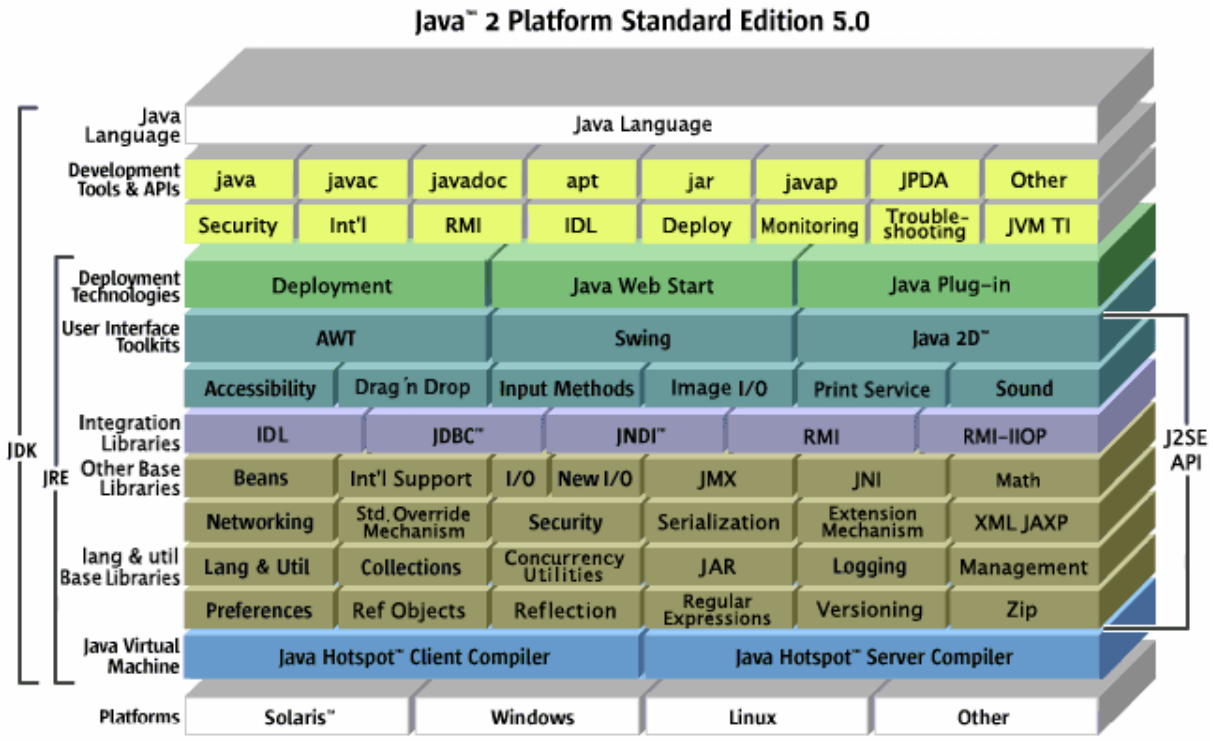
Sun Microsystems definierar Javas API som följer:

[174] Lindholm, Tim och Frank Yellin, *The JavaTM Virtual Machine Specification*.
[175] Harold, Elliotte Rusty, Hemligheterna i Java. (Jönköping: IDG AB, 1997)
[176] Ibid.
[177] Lindholm, Tim och Frank Yellin, *The JavaTM Virtual Machine Specification*

"The Java Application Programming Interface (API) is prewritten code, organized into packages of similar topics. For instance, the Applet and AWT packages include classes for creating fonts, menus, and buttons. The full Java API is included in the Java 2 Standard Edition download."[178]

Java API är alltså en uppsättning klasser som kompilerats till class-filer som JVM kan använda sig av. Förutom AWT som Sun Microsystems beskrivit finns även grafikbiblioteket Swing. Swingbiblioteket ger möjlighet att utveckla gränssnitt som är plattformsoberoende och som anropar de grafikinställningar som användaren har.[179]

För att JVM och Javaapplikationen ska kunna nyttja Javas API måste API:et med sina class-filer finnas installerat på den plattform som programmet ska exekveras på. För att Javas API-class-filer ska passa den plattform som de ska exekveras på är dessa filer plattformsspecifika, men för den exekverande applikationen ser gränssnittet mot API:et ut på samma sätt oavsett vilken plattform det körs på.[180]

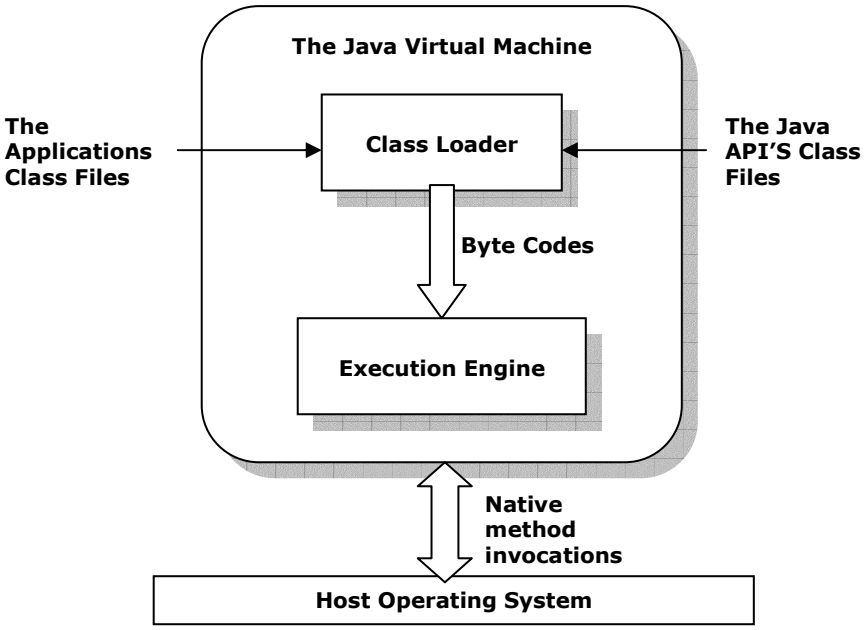


Figur 7 Översikt över Java 2 Platform Standard Edition 5 med dess API inkluderat.

Java native

Då JVM implementeras som mjukvara i ett operativsystem interagerar Javaprogrammen som körs i JVM med operativsystemet genom att anropa native-metoder enligt bilden nedan. En native-metod är skriven i ett annat språk, så som C eller C++, och kompileras till native maskinkod knuten till en viss processor.[181]

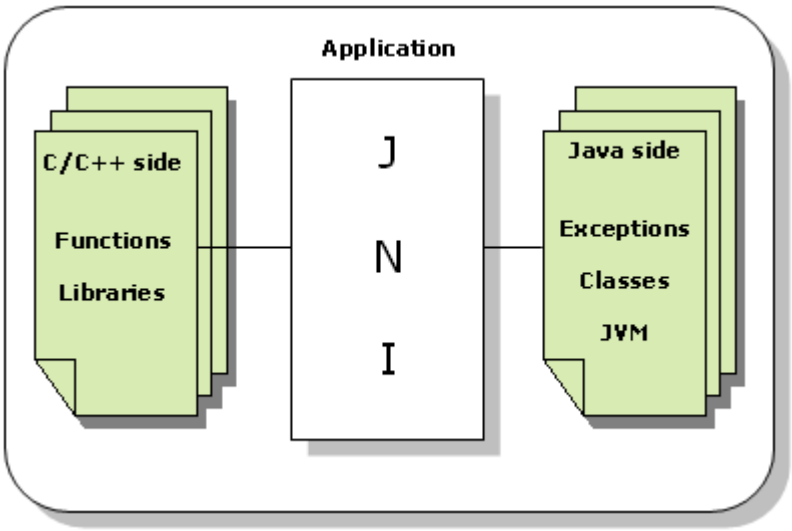
[178] Sun Microsystem, *New to Java Programming Center – Unraveling Java Terminology.*
[179] Venners, Bill, *Inside the Java Virtual Machine*.
[180] Ibid.
[181] Venners, Bill, *Inside the Java Virtual Machine*.



Figur 8 Javas virtuella maskin interagerar med operativsystemet genom native-metoder.

Native-metoder lagras i dynamiska bibliotek som till exempel dll-[182] och so-filer[183] vilka är plattformsspecifika. Då programmet exekveras och anropar en native-metod laddas det dynamiska biblioteket in i JVM och därefter anropas native-metoden i biblioteket.

Genom att använda native-metoder i Javaprogram går det att nyttja programkod som är skriven i C eller C++ men problemet med detta är att applikationen då blir plattformsspecifik eftersom C/C++-koden är knuten till ett visst operativsystem.[184] För att göra applikationen mindre plattformsberoende har Sun Microsystems skapat Java native interface (JNI). JNI fungerar som ett gränssnitt mellan native-koden och Javakoden och tillåter programkod skriven i olika språk att interagera med varandra genom anrop av varandras metoder.[185]



Figur 9 Översiktsbild över JNI

[182] Definition av *"Dynamic Link Library (dll)"*, se Bilaga 1.
[183] Definition av *"Shared Object Library (so)"*, se Bilaga 1.
[184] Venners, Bill, *Inside the Java Virtual Machine*.
[185] Campione, Mary et al, "Java Native Interface", *The Java Tutorial.*

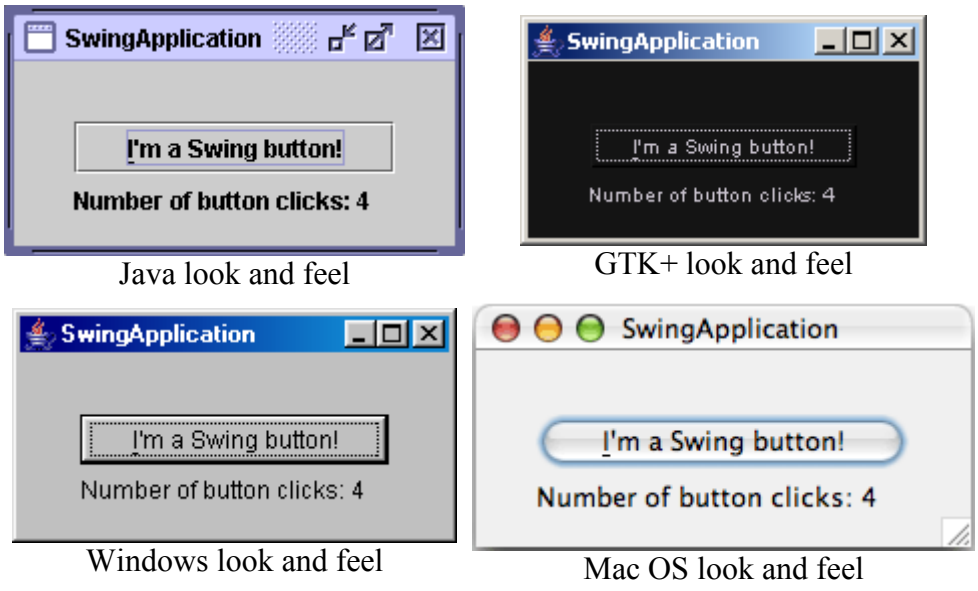
För att applikationen ska bli mindre plattformsoberoende krävs att den programkod som är skriven i C/C++ är plattformsoberoende i den mening att programkoden och de dynamiska bibliotek som skapas är kompilerade och genererade för de olika plattformar som applikationen ska exekveras på. Till exempel måste C/C++-kod som ska köras med JNI under Windows först modifieras med hjälp av JNI-kod så att den kan kommunicera med Javakod. Den måste sedan kompileras och paketeras i en dll-fil. Om samma kod ska köras under Linux kompileras koden istället under Linux och paketeras i en so-fil. JNI tillåter att plattformsoberoende anrop görs och beroende på operativsystem laddas lämplig dll eller so-fil.[186]

Garbage Collector

Javas har en så kallad Garbage Collector, en skräpsamlare, som frigör minne som utnyttjas av objekt som inte längre används, det vill säga, inte längre har någon referens. Detta sker automatiskt, och det enklaste är oftast att låta skräpsamlaren ta hand om uppgiften, men ibland kan man ha nytta av själv styra händelsen på grund av att man i nästkommande fas kommer att kräva extra minnesutrymme, eller efter att man utfört en handling som genererar flera orefererade objekt som inte längre ska användas. Denna handling görs genom att programmeraren själv anropar metod för skräpsamlaren i System-klassen.[187]

Swing

Javas grunduppsättning av bibliotek innefattar bland annat ett bibliotek kallat Swing med hjälp av vilket man kan bygga grafiska gränssnitt som är ämnade att användas på ett flertal plattformar. Detta bibliotek tillhandahåller också möjlighet att få ett plattformsspecifikt utseende, beroende av vilken plattform som applikationen exekveras på.[188] För att undvika att få ett oönskat resultat vid visning av gränssnittet rekommenderas att man använder sig av en layoutmanager istället för att specificera komponenters placering. Dessutom bör man undvika antaganden om sådant som typsnitt-storlekar och skärmstorlek och upplösning.[189]



Figur 10 Look and feel för olika plattformar med hjälp av Javas Swing-bibliotek

[186] Campione, Mary et al, "Java Native Interface", *The Java Tutorial.*
[187] Sun Microsystems, *The Java Tutorial*.
[188] Sun Microsystems, *Java Look and Feel Design Guidelines.*
[189] Ryall, Ken, *Writing Java on the Mac for All Platforms*, 2004.

Olika egenskaper hos operativsystem

För att göra ett program plattformsoberoende krävs att hänsyn tas till de övergripande olikheter som finns hos olika operativsystem. Nedan visas på ett antal olikheter mellan Windows, Mac OS och Linux.

Big- och Little-endian

Då data består av fler än en `byte` kan detta lagras i datorns minne på två olika sätt, antingen enligt big-endian-systemet eller enligt little-endian-systemet. Då data lagras enligt big-endian-systemet så lagras den mest signifikanta `byten`[190] i sekvensen på den lägsta lageradressen. I little-endian-systemet lagras istället den minst signifikanta `byten`[191] i sekvensen på den lägsta lageradressen. Nedan följer ett exempel på dessa två system då talet 1025 lagras i en 4 `byte` stor `integer`:[192]

Binär representation av talet 1025: `00000000 00000000 00000100 00000001`

ADRESS	BIG-ENDIAN	LITTLE-ENDIAN
00	00000000	00000001
01	00000000	00000100
02	00000100	00000000
03	00000001	00000000

Tabell 7 Big- och little-endian-tolkningar av talet 1025

De flesta moderna datorer använder sig av little-endian-systemet. Mac använder PowerPC-arkitekturen som är bi-endian-system och förstår båda systemen och Java använder sig uteslutande av big-endian-systemet.[193,194,195]

Vid migrering mellan Windows och Mac har problem påträffats då filer utbytts mellan de olika operativsystemen och där en fil sparats enligt ett system och laddas med ett annat. Byte-ordningen har då blivit feltolkad.[196]

Filhantering

Det största arbetet vid migrering av kod är att ta hänsyn till operativsystemens olika filsystem.[197] Nedan följer några av de skillnader som finns.

Filsystemsstruktur

Gemensamt för filstrukturerna i Windows, Mac OS och Linux är den hierarkiska uppbyggnaden. Operativsystemen har olika namn för sina mappar, eller kataloger men strukturen är densamma. Filsystemet utgår från en rot under vilka underkataloger finns lagrade. Dessa underkataloger innehåller sedan ytterligare underkataloger och filer vilka bygger upp den hierarkiska strukturen.

---

[190] Definition av *"Mest signifikanta byten"*, se Bilaga 1.
[191] Definition av *"Minst signifikanta byten"*, se Bilaga 1.
[192] Webopedia.com, *Online Computer Dictionary for Computer and Internet Terms and Definitions.*
[193] Ibid.
[194] Webopedia.com, *Online Computer Dictionary for Computer and Internet Terms and Definitions.*
[195] Canadian Mind Products, *Java Glossary.*
[196] Mindfire Solutions, *Mac OS-Windows Porting*.
[197] Microsoft, *From One Code Base to Many Platforms Using Visual C++*.

Roten i Windows betecknas med C:\, vilket normalt motsvarar den primära hårddisken. Under roten lagras mappar, som i sin tur kan innehålla ytterligare mappar som innehåller filer eller program. Exempel på kataloger för Windows:[198]

KATALOG	INNEHÅLL
C:\Windows	Konfigurationsfiler för Windows-miljön, tillbehör mm.
C:\System	Vanliga dll-filer, systemkonfiguration.
C:\Programs	Länkar för exekvering av program.

Tabell 8 Kataloger i Windows

Roten i Linux betecknas med tecknet slash (/). Under roten återfinns ett flertal underkataloger, som /dev, /boot, /bin, /lib och så vidare. Under till exempel /home-katalogen finns användarnas utrymme för lagring. Varje användare av filstrukturen får en egen katalog som då betecknas med /home/minkatalog. En fördel med Linux filstruktur är att användaren vanligtvis inte behöver känna till vilken disk informationen ligger lagrad på, bara sökvägen. Exempel på kataloger för Linux:[199]

KATALOG	INNEHÅLL
/root	Rotanvändarens lagringsutrymme.
/home	Kataloger för användarnas lagringsutrymme.
/bin	Vanliga kommandon, t ex ls, cp, mv.
/usr	Linuxprogram, bibliotek och delad data.
/mnt	Kataloger för extern hårdvara, floppy, cd-rom mm.

Tabell 9 Kataloger i Linux

Mac OS har samma trädliknande filstruktur som de ovan nämnda operativsystemen, och har en ”desktop”-fil som motsvarande roten i Windows och Linux.[200,201] Det som egentligen skiljer Mac OS från de andra i det här avseendet är Macintoshs filer har två delar, eller forks, som data- respektive resourcefork. En datafork lagrar, som namnet avslöjar, data som filen innehåller, medan resourcefork håller en hel mängd information, såsom vilken ikon filen använder, vilken applikation som skapat den och mycket mer.[202] Exempel på kataloger för Mac OS:[203]

[198] The Shell, *Wine User Guide.*
[199] Codemartial, *The Structure of Linux*, 2005.
[200] Dartmouth, *Storing information – The file system,* 2004.
[201] Express Design Inc, “*Unixguide – File System Layout*”.
[202] Office of Information Technology, *Mac: File system,* 2002.
[203] Chapman, Lindell, *File systems*, 2004.

Katalog	Innehåll
Applications	Program
System Folder	Systemfiler
Library	Typsnitt mm
Users	Användarkonton
Documents	Dokument

Tabell 10 Kataloger i Mac OS

Katalogseparatorer och filnamn

I Windows skrivs en sökväg till en fil med tecknet för backslash (\) som katalogseparator. Exempelvis C:\WINDOWS. I Linux används istället tecknet för slash (/) och i Mac OS 9 används tecknet för kolon (:).[204] I Mac OS X samt hos senare versioner av Mac OS används, precis som i Linux, slash (/) som katalogseparator.[205]

Då en fil ska sparas i ett operativsystem finns ofta restriktioner för hur filnamnet får se ut och otillåtna tecken som inte får användas. Då en fil sparas i Windows får filnamnet inte bestå av följande tecken "/*?<>|: i Mac OS 9 är det bara kolontecknet som inte får finnas med i filnamnet medan det i Mac OS X är både kolon och slash. I Linux är slash ett otillåtet tecken.[206,207,208,209] Rekommenderat är också att man undviker att använda space, det vill säga mellanslag.[210] För att vara på den säkra sidan när man namnger filer som ska användas på ett flertal plattformar är bör man bara använda sig av bokstäver, siffrorna 0-9 samt tecknen _, $, ~, !, #, - , (,), och ’.[211]

Filassociation

I Windows används filändelser för att identifiera filtypen och associera denna till en applikation och dess ikon. På grund av detta kommer en och samma applikation associeras till en viss filtyp, och den associerade applikationen kommer att användas för att visa den, oavsett vilken applikation som använts för att skapa den. Mac OS filer lagrar mer information om filen i anslutningen till den, såsom information om filen och dess ägare, som beskrivits i tidigare avsnitt. Filen identifieras på så vis med hjälp av fler variabler än enbart filändelsen. På så sätt kan filtyper med samma ändelse associeras med olika applikationer.[212] I Linux ses allt som inte är en process, som en fil. Även kataloger ses som filer eftersom även de är filer, innehållande namn på de filer som finns i dem. Varje fil i Linux har tillgång till en så kallad inode. Den innehåller en mängd information om filen, bland annat om filens ägare, typ,

[204] Comentum, *Comparison of File and Directory Name and File Systems on Macintosh, PC and Unix Computers and Operating systems*.
[205] Lewis & Clark College, *Getting started with OS X.*
[206] Comentum, *Comparison of File and Directory Name and File Systems on Macintosh, PC and Unix Computers and Operating systems*.
[207] Lewis & Clark College, *Getting started with OS X.*
[208] Comentum, *Comparison of File and Directory Name and File Systems on Macintosh, PC and Unix Computers and Operating systems*.
[209] Lewis & Clark College, *Getting started with OS X.*
[210] About.com, *Focus on Linux*, 2005.
[211] Fleming, R. Mark, *Cross-platform compatible files – File Naming Issues,* 2002.
[212] Doughty, Mike, *Understanding Macintosh and Windows File Types*, 2003.

rättigheter, datum då den skapats, öppnats eller förändrats, antal länkar till filen, storlek och lagringsplats. En inode håller också information om när den själv blivit uppdaterad. Den enda informationen den inte håller om filen är filens namn och katalog. Denna information lagras i speciella katalogfiler, som tillsammans med rätt inode, ger fullständig information om filen.[213]

Sökvägslängd

De flesta Windows och Linux-versioner har stöd för 255 tecken i filnamnet och Windows har även stöd för 260 tecken i en sökväg. Mac OS 9 har bara stöd för 31 tecken i filnamnet men efter Mac OS X har även det operativsystemet stöd för 255 tecken.[214]

Skiftlägeskänslighet

Med skiftlägeskänslighet (engelskt ord: Case Sensitive) menas i detta fall om operativsystemet skiljer mellan bokstäver som skrivs med versaler eller gemener. Ett skiftlägeskänsligt operativsystem skulle se orden Dator och dator som två olika ord medan ett icke-skiftlägeskänsligt operativsystem skulle se de som samma ord.[215] Av Windows, Mac OS och Linux är Linux det enda operativsystemet som är skiftlägeskänsligt.

Nedan ses en tabell som på ett överskådligt sätt visar skillnader i filhantering i de olika operativsystemen:

KATEGORI/OS	WIN 95/98	WIN 2000+	MAC OS 9	MAC OS X	LINUX
Max. filnamnslängd	255	256	31	255	255
Max. sökvägslängd	260	260			
Otillåtna tecken	"/*?<>|:	"/*?<>|:	:	: /	/
Katalogseparator	\	\	:	/	/
Skiftlägeskänsligt	NEJ	NEJ	NEJ	NEJ	JA

Tabell 11 Jämförelse av filhantering i Windows, Linux och Mac OS.[216,217]

Inställningar för system och användare

Under Windows har man tillgång till Windows register, som lagrar information om en PC i form av bland annat inställningar för hårdvara, mjukvara och användare. Detta register, vilket innefattar ett fåtal filer med lagrad data, är specifikt för Windows operativsystem. En direkt motsvarighet till registret saknas i andra operativsystem. Mac OS lagrar motsvarande uppgifter under användarkatalogen i NetInfo, som lagrar administrations- och installationsuppgifter,[218] och Defaults System,[219] som lagrar uppgifter om applikationer och

[213] Linux Forum, *General overview of the Linux file system,* 2003.
[214] Comentum, *Comparison of File and Directory Name and File Systems on Macintosh, PC and Unix Computers and Operating systems*.
[215] Computer Hope, *Case Sensitive*.
[216] Comentum, *Comparison of File and Directory Name and File Systems on Macintosh, PC and Unix Computers and Operating systems*.
[217] Lewis & Clark College, *Getting started with OS X.*
[218] Graham, Alan, *Home on the Go with NetInfo.*
[219] Developer Connection, *BSD General Commands Manual.*

användare.[220] I Linux lagras motsvarande uppgifter i /etc respektive /home eller i konfigurationsfiler.[221]

Plattformsoberoende

Användargränssnitt

Eftersom C och C++ används av så många är flertalet verktyg för grafiska gränssnitt utvecklade för att användas av dessa språk, men vissa verktyg, Fresco till exempel, försöker vara språkneutrala. Sådana verktyg får man tillgång till genom bibliotek, som man måste kompilera och länka för varje system de ska användas på. Det finns också verktyg som Java, Tcl och Smalltalk som interpreteras och därför inte behöver kompileras för varje plattform.[222]

Filformat

Det finns en mängd filformat att utnyttja i en plattformsoberoende applikation.[223,224,225,226] Filer av bland annat dessa format kan utan problem användas på ett flertal plattformar:

FILTYP	FILÄNDELSE	FILFORMAT
Publiceringsfiler	TIF (TIFF)	Tagged Image File Format
	EPS	Encapsulated PostScript
	PDF	Portable Document Format
Text	TXT	Text
	RTF	Rich Text File
	DOC	Document
Web	HTM (HTML)	HyperText Markup Language
	XML	Extensible Markup Language
	SWF	Macromedia Flash File Format
Bild	GIF	Graphics Interchange Format
	JPG (JPEG)	Joint Photographic Experts Group
	PNG	Portable Network Graphics

Tabell 12 Plattformsoberoende filformat

[220] Runrev.com, *OT: Mac OS X have a registry?*
[221] Shields, Ian, "*Linux SEK 2005 Release 1: Basic tasks for new Linux developers*", 2005.
[222] Babcock, Michael, *The Importance of the GUI in Cross Platform Development*, 1998 .
[223] Hokum Glossaries *File extensions,* 2004.
[224] Chopra, Manish, *Cross-Platform Computing*, 2002.
[225] Imagemontage, *Cross-Platform Compatible Files,* 2002.
[226] Quark.com, *tech-notes Cross-Platform Issues: graphics*, 2005.

Typsnitt

Här följer en figur som visar på typsnitt som finns tillgängliga för olika typer av plattformar. De gröna bockarna indikerar vanligt förekommande typsnitt och de gula bockarna indikerar Microsoft webbtypsnitt som inte alltid följer med alla operativsystem.[227]

Generic	Font	Windows 9x	Windows 2000/XP	Mac Classic	Mac OS X	Linux Unix
serif	Times New Roman	✓	✓	✓	✓	✓
	Times			✓	✓	✓
	Georgia	✓	✓	✓	✓	✓
sans-serif	Andale Mono	✓	✓	✓	✓	✓
	Arial	✓	✓	✓	✓	✓
	Arial Black	✓	✓	✓	✓	✓
	Helvetica			✓	✓	✓
	Impact	✓	✓	✓	✓	✓
	Trebuchet MS	✓	✓	✓	✓	✓
	Verdana	✓	✓	✓	✓	✓
cursive	Comic Sans MS	✓	✓	✓	✓	✓
fantasy						
monospace	Courier New	✓	✓	✓	✓	✓
	Courier			✓	✓	✓

Figur 11 Typsnitt tillgängliga för olika operativsystem

Av typsnitten i figuren finns det några som är mycket lika varandra och de kan därför användas som ersättare för varandra i de olika operativsystemen, här visas en figur över dem:[228]

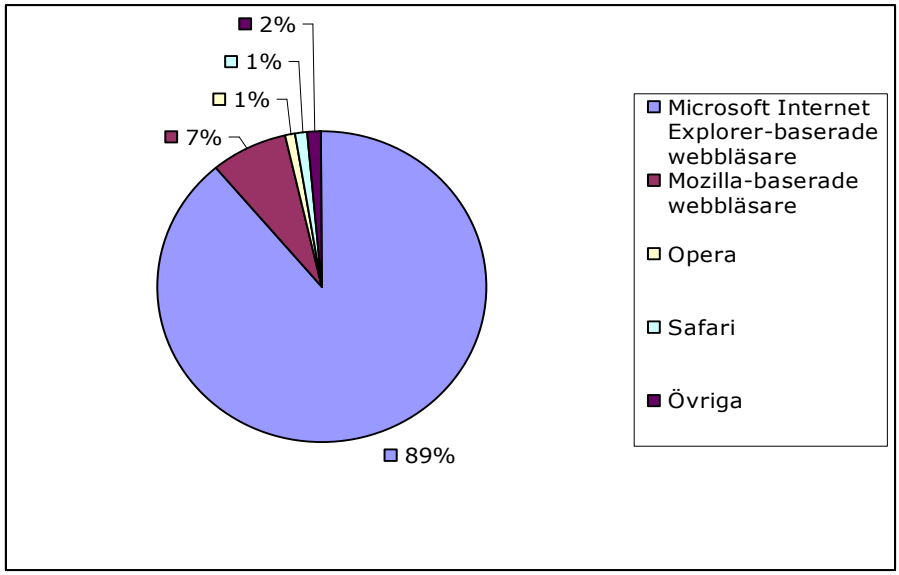
Generic	Similar Fonts
serif	Palatino, Palatino Linotype, Book Antiqua
	Times, TimesNR, Times New Roman
sans-serif	Arial, Helvetica
	Arial Rounded, VAG Rounded
	Avant Garde, Century Gothic
	Optima, Omega, Zapf Humanist
	Univers, Zurich
monospace	Courier, Courier New

Figur 12 Liknande typsnitt

Webbläsare

Idag finns det en mängd olika webbläsare ute på marknaden.[229] Statistik från november 2004 visar fördelningen av användare av de mest populära webbläsarna:[230]

[227] Browser News, Resources - Fonts, 2005.
[228] Ibid.
[229] Wikipedia the Free Encyclopedia. *List of web browsers.*
[230] Wikipedia the Free Encyclopedia. *Web browser.*



Figur 13 Användarfördelning för webbläsare

Ur diagrammet kan utläsas att Microsoft Internet Explorer tillsammans med webbläsare som är baserade på densamma är överlägset den mest använda läsaren på marknaden. Internet Explorer finns tillgänglig för Windows och Mac OS men inte Linux. Mac OS-versionen av Internet Explorer slutade utvecklas efter version 5.[231]

Många webbläsare går utan problem att använda på ett flertal plattformar. När man som utvecklare använder sig av dem i en applikation, bör man därför endast utnyttja grundfunktionerna för att vara säker på att resultatet verkligen blir plattformsoberoende, dock på bekostnad av att man förlorar den senaste funktionaliteten. [232]

The World Web Consortium (W3C) skapar internationell standard och specifikationer för hur en webbläsare ska fungera för att vara plattformsoberoende. W3C publicerar också standard för HTML-dokument och taggar. Om alla utvecklare av webbläsare hade följt dessa standarder skulle det därför inte vara några problem med att använda dem på olika plattformar. [233]

Det finns dock anledningar till varför inte dessa följs fullständigt. Standarder uppdateras ständigt, vilket gör att webbläsarna i så fall måste uppdateras i samma takt. Dessutom konkurrerar webbläsarna med varandra och försöker därför skapa ny funktionalitet. Webbläsarna har dock en förmåga att följa varandra i utveckling, och nya funktioner i en webbläsare appliceras efterhand på fler webbläsare. Det kan därför vara värt att testa funktionerna i de olika webbläsarna om man är osäker på att man får ett plattformsoberoende resultat.[234]

[231] Microsoft, *Internet Explorer Home.*
[232] Knight, D et al, *Platform Independence – Final report*, 2001.
[233] Ibid.
[234] Ibid.

Resultat och diskussion

I detta kapitel presenteras de problem som framkommit under det tre månader långa etnografiska experimentet som ligger bakom studien. Kapitlet innehåller en presentation av de problemkategorier som tagits fram samt alla de enskilda problem som var och en av kategorierna innehåller. I resultatet har även lagts till diskussion och rekommendation kring problem, med grund i erfarenhet experimentet givit, som kan vara till nytta för praktiker. Inledningsvis beskrivs omständigheterna kring det experiment som utförts.

Experimentets upplägg

För att undersöka vilka problem som kan uppstå vid migrering för att uppnå plattformsoberoende har ett experiment gjorts inom ett företag. Experimentet bestod i att migrera en Windows-applikation till en plattformsoberoende applikation. Företaget som experimentet utförts inom grundades 2003 och har sitt huvudkontor i Göteborg. De utvecklar och för produkter med vilka man kan skicka och ta emot krypterad e-post. De har även en gratisprodukt med vars hjälp man kan läsa krypterad e-post. I takt med att betalprodukterna blivit allt mer populära har användare börjat efterfråga gratisprodukten för andra operativsystem såsom Mac OS och Linux, detta för att de ska kunna skicka krypterad e-post även till dem som nyttjar andra plattformar. Några av betalprodukternas kunder är företag vilka använder produkterna för att skicka ut användarnamn, lösenord och annan känslig information till sina kunder. För dem innebär det en begränsning att kunderna endast kan läsa den krypterade e-posten på Windows. En lösning på efterfrågan om att kunna läsa krypterad e-post även på andra operativsystem än Windows är att skriva om gratisprodukten i Java.

I dagsläget kan man använda gratisprodukten interagerad i Microsoft Outlook eller som en helt fristående applikation. Gratisprodukten fungerar på följande sätt: Mottagaren får e-post på vanligt sätt i sin e-postklient eller till sin webbmail. Som bifogad fil i e-posten medföljer det krypterade meddelandet. Den krypterade filen har filändelsen .sema. Vid programmets installation registreras denna filändelse i Windows och associeras samtidigt till produkten. Detta medför att mottagaren kan spara sema-filen på hårddisken och öppna denna genom att dubbelklicka på den eller genom att öppna gratisprodukten och där välja filen. För att den krypterade sema-filen ska öppnas krävs att mottagaren anger lösenordet som denne och sändaren av det krypterade meddelandet kommit överrens om. Enkelt uttryckt hanterar programmet kryptering, dekryptering och organisering av sema-filer. Det hanterar inte sändningen och mottagandet av e-post.

För studiens räkning och dess genomförande gjordes den befintliga programvarans källkod tillgänglig. Källkoden, inklusive gränssnittet är skriven i programspråket C++[235] och utnyttjar biblioteket Microsoft Foundation Class Library (MFC).[236] Källkoden använder sig också av två externa bibliotek. Ett av dem är Synchronous Key Generator (SKG) vilken är till för att generera de nycklar som används vid kryptering och dekryptering av sema-filen. Det andra externa biblioteket är Crypto++ vilket är ett bibliotek som används för kryptering och dekryptering, samt för att göra programmet och koden i stort säkert. SKG:n finns för Java, däremot är Crypto++ enbart tillgängligt för C++.

Det fanns ingen dokumentation att tillgå för att få en bra förståelse för programkoden och de ingående projektens struktur. För studiens genomförande har diskussioner med personalen på

[235] Definition av *"C++"*, se Bilaga 1.
[236] Definition av *"Microsoft Foundation Class Library"*, se Bilaga 1.

företaget samt diverse reverse-engineering-verktyg, böcker och Internetsidor varit till stor hjälp. Experimentets övergripande strategi har varit förståelse för den befintliga källkoden, identifiering av de viktiga funktionerna, och till sist, översättning till Java.

För att utföra experimentet avsattes plats för detta på företaget som utvecklat applikationen som skulle migreras. Även resurser i form av datorer och verktyg gjordes tillgängliga för experimentet. Experimentet tvingade till ett förhållningssätt där två roller utnyttjades. Migreringen utfördes och observerades parallellt av oss, på platsen, där det alltid fanns möjlighet att få stöd från de utvecklare som programmerat den ursprungliga applikationen.

Presentation av problematik

De problem som påträffats har enligt Grounded theory sammanställts under olika kategorier, vilka bland annat syftar till att ge en mer greppbar bild över de stora problemområdena. Studien har lett fram till de fyra problemkategorier som redovisas nedan.

1. Problematik rörande startfasen
2. Problematik rörande programspråkens skillnader
3. Problematik rörande datalagring
4. Problematik rörande externa bibliotek och komponenter

Nedan följer en beskrivning av var och en av de ovanstående kategorierna, deras ingående problem och en presentation av erfarenheter som införskaffats under denna studie samt rekommendationer på eventuella lösningar på problem.

1: Problematik rörande startfasen

Under denna kategori har samlats de problem som har egenskapen av att vara en del av startfasen av migreringen. De karaktäriseras av att de inte ingår i själva arbetet med överföringen av kod, men de är problem som har stötts på i det nödvändiga förarbetet som måste genomföras innan den verkliga migreringen kan ske. Det handlar bland annat om den ursprungliga applikationens design, men även om utvecklingsverktyg.

Avsaknaden av dokumentation (**problem 1.1**) gjorde att startfasen blev betydligt mer tidskrävande än väntat. De dokument som fanns tillgängliga var snarare utvecklade i marknadsföringssyfte än som beskrivande designdokument.

Ett försök till reverse engineering utfördes därför för att få en överblick över applikationen. Denna process gav inte det önskade resultatet på grund av att applikationen gjorde anrop till dll-filer (**problem 1.2**) som innehöll ytterligare applikationskod. Därför kunde inte källkoden presenteras grafiskt i sin helhet. Detta gav till följd att en helt manuell genomgång av en omfattande mängd kod (**problem 1.3**) fick göras.

Det som ytterligare försvårade uppgiften var att endast en del av den fullständiga applikationen skulle migreras, varför det också var nödvändigt att utreda vilken del av källkoden som hörde till vilken del av applikationen (**problem 1.4**). Källkodens modularitet var professionellt uppbyggd, men gjorde det därför också svårt att följa de olika metodanropen och utreda dess funktionalitet (**problem 1.5**).

Som tidigare nämnts utnyttjade applikationen ett antal externa komponenter. För att kunna följa den ursprungliga källkoden var det nödvändigt att tillhandahålla kunskap om programspråk, bibliotek och externa komponenter (**problem 1.6**).

Eftersom syftet var att utveckla en plattformsoberoende produkt möttes inga hinder att utveckla den nya versionen av applikationen på den ursprungliga plattformen – Windows.
När det var dags att välja verktyg för den vidare utvecklingen uppdagades dock att det ursprungliga verktyget för utveckling, Microsofts Visual Studio .NET 2003 inte motsvarade kraven för migreringsutvecklingen (**problem 1.7**). Programmet utnyttjades för källkodsrevidering och debuggning men gav inte möjlighet att utnyttjas för utveckling i Java.

PROBLEM #	BESKRIVNING
Problem 1.1	Designdokument för applikationen saknades.
Problem 1.2	Huvudapplikationen anropade dll-filer som innehöll mer applikationskod vilket försvårade reverse engineering.
Problem 1.3	Applikationen innehöll en omfattande mängd kod.
Problem 1.4	Det var bara en del av den ursprungliga applikationens kod som skulle migreras.
Problem 1.5	Applikationskodens modulära uppbyggnad och många anrop var tvungna att följas med hjälp av debuggning.
Problem 1.6	En omfattande mängd förkunskaper krävdes för förståelse av källkoden.
Problem 1.7	Det ursprungliga verktyget för utveckling, Microsofts Visual Studio .NET 2003, motsvarade inte kraven för migreringsutvecklingen.

Tabell 13 Problematik rörande startfasen

Diskussion och rekommendation kring problematik med startfasen

En migrering kräver en förståelse för den ursprungliga applikationen och det rekommenderas därför att man tar hjälp av dokumentation, reverse engineering eller enkla diagram för att få en övergripande bild av struktur, omfattning, beroenden och funktionalitet.[237] Denna startfas blev, av skäl som belyses nedan, en långt mer tidskrävande process än väntat, vilket till stor del orsakats av just avsaknad av stöd i form av dokumentation för applikationen och dess beståndsdelar.

På grund av avsaknaden av dokumentation, gjordes därför ett försök med reverse engineering. Då applikationen anropade dll-filer, innehållande ytterligare programkod, uppstod svårigheter med att få en tydlig bild av applikationen som helhet. Därför startades en manuell genomgång av den omfattande mängd rader kod applikationen bestod av. Förhoppningen var att reda ut vilka kodrader som var av betydelse för den del av applikationen som skulle migreras, hur arkitekturen var byggd, hur modulerna samverkade, samt vilka beroenden som fanns mellan källkodskärnan, grafiska gränssnitt samt externa bibliotek och komponenter.
Källkoden var modulärt uppbyggd och innehöll en omfattande mängd metodanrop. Det gav struktur åt koden, men gjorde den samtidigt komplicerad att följa. Att utreda metodernas funktionalitet fick därför göras genom debuggning vilket blev ytterligare ett utdraget steg i fasen.

[237] Mindfiresolutions, *Porting: A development primer*.

Applikationens var, som tidigare nämnts, beroende av ett antal externa bibliotek och komponenter. Därför krävdes, för förståelse av källkoden, kunskap om såväl programspråket C++, men också kunskap om biblioteken Microsoft Foundation Class Library (MFC), säkerhetsbiblioteket Crypto++, den externa applikationen Synchronous Key Generator (SKG), samt kunskaper om språk för webbdokument såsom HTML och XML.

Migreringens egenskaper i form av flertalet målplattformar gjorde det möjligt att utveckla den nya versionen av applikationen under samma plattform som den ursprungliga, det vill säga under Windows. Verktyget som ursprungligen använts, Microsofts Visual Studio .NET 2003 utnyttjades för källkodsrevidering och debuggning, men gav inte stöd för Java. Valet av utvecklingsverktyg föll därför på Eclipse SDK 3.0.1 vilket också gav möjlighet att utnyttja befintligt versionshanteringsprogram, integrera Javas API, samt Javas Swing-bibliotek och komponenter för att kunna bygga gränssnitt.

Hur komplicerad den ursprungliga applikationens källkod och struktur är, är inget man kan påverka. Att följa allmänna riktlinjer för migrering kan däremot underlätta arbetet.

- Skapa en förståelse för den ursprungliga applikationen
- Hitta resurser för stöd i arbetet, dokumentation, andra utvecklare, forum
- Skaffa kunskap om de programspråk, bibliotek, komponenter och plattformar som migreringen kommer att beröra.
- Skaffa information och kunskap om de utvecklingsverktyg och miljöer passande för migreringen, använd om möjligt en miljö som är kompatibel med både ursprungs- och målplattformen.

2: Problematik rörande programspråkens skillnader

Denna kategori har ett stort antal ingående problem och många av dem är väldokumenterade i litteraturen. Det som karaktäriserar problemen är att de berör skillnader mellan C++ och Java. Även om programspråken uppfattas som lika och att de olikheter som finns inte påverkar i någon större grad så kan alla problem tillsammans förhindra arbetet och ta tid. Problemen har dykt upp frekvent under hela experimentet då studien befunnit sig i kodmigreringsfasen. Vissa problem har varit lätta att hitta lösningar på medan andra har varit mer tidskrävande. Problem som påträffats under detta experiment har bland annat att göra med datatypernas storlek och det tillsammans med att `unsigned` inte finns i Java har inneburit en del tidskrävande problem.

Vid migreringen av en klass från den existerande källkoden till Java har många av de problem som tas upp i litteraturen påträffats. Davis tar upp skillnader mellan programspråken där han bland annat visar på hur hela klasstrukturen måste förändras vid migrering till Java, där två filer ska sättas ihop till en fil och alla variabler och metoders tillgänglighet läggas till. En klass i C++ består av två filer, en h-fil och en cpp-fil som tillsammans bygger upp klassen.[238] Detta var ett problem då en klass i Java bara består av en fil, med ändelsen java (**problem 2.1**). Ett annat problem som noterades var att klasstrukturen i C++ och Java skilde sig markant åt. Dels skiljer sig klasstrukturen åt beroende av filuppdelningen i C++, där h-filen fungerar som en beskrivning av cpp-filen. Dels skiljer sig klasstrukturen i sättet på vilket de olika variablerna deklarerades (**problem 2.2**). I C++ deklareras alla variablers åtkomst under ett reserverat ord som bestämmer dess tillgänglighet. I Java deklareras åtkomst framför varje variabel. Även arvssyntaxen skilde åt, där C++ deklarerar ett arv med kolon, `:`, deklarerar Java det med det reserverade ordet `extends` (**problem 2.3**).

[238] Davis, Mark, *Porting C++ to Java.*

Problematik kring skillnader mellan typer i Java och C++ var en av de punkter som skiljde programspråken åt och som medfört flest problem. Davis tar upp detta och beskriver bland annat att `unsigned` inte finns i Java, där alla variabler utom `char` är fördeklarerade som `signed`.[239] Genomgående i den befintliga applikationens källkod används variabler som deklarerats som `unsigned`, prefixet som inte finns i Java (**problem 2.4**). Främst är det `byte` och `char` som deklarerats med detta prefix. En `byte` innehar självklart storleken av en `byte` i både Java och C++, men `char` däremot har storleken av en `byte` i C++ och två `byte` i Java så även om Javas `char` är `unsigned` uppstår problem (**problem 2.5**). I källkoden användes `byte` och `char` flitigt och oftast användes variabeltyperna i samband med värdekritiska funktioner som var relaterade till säkerhet, kryptering och dekryptering.

I några av de funktioner som var nödvändiga att migrera arbetade koden på bitnivå på `char`-variabler som var deklarerade som `unsigned`. De funktioner som använde sig av bitoperator-funktionerna var beroende av att `char`-arrayen som deklarerades som privat medlemsvariabel innehöll en `byte` stora värden. Detta gjorde att `char`-arrayen fick deklareras som en `byte`-array i Javakoden, vilket förtydligar problem 2.5. Problemet som uppstod då var att Javas `byte` är `signed` och alltså innehåller både negativa och positiva tal. Så i C++ koden fanns en `char`-array där varje `char` kunde innehålla värden mellan 0 och 255 och i Javakoden en `byte`-array där varje `byte` kunde innehålla värden från -128 till 127. En situation som understryker problem 2.4 uppstod. Davis beskriver hur detta ställer till problem då migrering mellan programspråken sker, då en variabel är deklarerad som `unsigned` i C++ fungerar inte Javas vanliga höger-shift-operator,[240] detta blev ett problem (**problem 2.6**). Vid detta problem rekommenderar Davis att man använder sig av Javas logiska höger-shift (>>>).[241] Vid användandet av logiska höger-shift-operatorer i Java och tal som är mindre än en integer (`int`) så konverteras talet som skickas in till en `int` och de bitar som läggs till för att utöka talet till en `int` sätts till 1. Shift-operationen utförs sedan och 0:or fylls på från höger enligt den logiska höger-shiftens regler. Efter detta konverteras talet tillbaka till den ursprungliga typen.[242,243] Detta vållade problem eftersom konvertering och påfyllningen med 1:or ofta förvandlade det shiftade talet till ett helt annat än det som avsågs (**problem 2.7**).

Davis beskriver att C++ preprocessor-kommandon inte fungerar i Java och det reserverade ordet `const` finns i Java.[244] Många klasser som ingick i källkoden använde sig av konstanter som fördefinierats med preprocessor-kommandot `#define`, varvid problem med migreringen uppstod (**problem 2.8**). Konstanter användes i C++ och deklarerades med hjälp av det reserverade ordet `const`, detta definierades också som ett problem (**problem 2.9**).

Fler problem som Davis tar upp är att Java inte har några referenser, att det inte går att använda defaultparametrar och att det inte finns några pekare.[245] Dessa problem stöttes också på och noterades under experimentets pågående. Referenser användes flitigt i koden och ändringar fick göras för att det skulle fungera i Java (**problem 2.10**). Även om det var sällan

[239] Davis, Mark, *Porting C++ to Java.*
[240] Ibid.
[241] Ibid.
[242] Wagner, Niel, *The Laws of Cryptography: Unsigned bytes in Java*, 2001.
[243] Eckel, Bruce, *Thinking in Java*. (Prentice-Hall, 2002).
[244] Davis, Mark, *Porting C++ to Java,* 1997.
[245] Ibid.

defaultparametrar användes i källkoden så berörde de migreringen och medverkade till problematiken (**problem 2.11**). I stora delar av den befintliga källkoden användes pekare, vilket inte finns i Java (**problem 2.12**). På vissa ställen användes också +-operatorn för att nå fram till ett visst ställe i pekarens innehåll detta gjorde att det behövdes en liknande funktion i Java för att komma fram till det specifika stället i det objekt som skapats i Java (**problem 2.13**).

I Java så är datatypen `String` oföränderlig, det vill säga dess innehåll kan aldrig förändras. Om ett stängobjekt skapats och tilldelats ett värde och detta värde behöver förändras kommer ett nytt objekt att skapas och det gamla värdet mister sin referens och kommer så småningom att samlas upp av Garbage collector.[246] I den befintliga applikationen hanterades i vissa fall lösenord eller nycklar, så fort dessa använts raderades de från minnet genom att de skrevs över med oväsentlig data. Om samma hantering skett i Java skulle strängen ändå ha varit kvar i minnet och en ny sträng med betydelselös data skulle ha skapats, stängen med lösenordet eller nyckel hade alltså inte raderats (**problem 2.14**).

Ett annat problem som uppstod handlade om hur de olika operativsystemen hanterar filer i fråga om little- och big-endian-systemen. Som beskrivits tidigare är Javaapplikationens syfte att dekryptera filer och att presentera innehållet. De filer som ska dekrypteras är skapade under Windows och därmed med little-endian-systemet. [247] Det finns därför en risk att Java, som nyttjar big-endian-systemet misstolkar informationen i filerna.[248] För att inläsningen ska fungera korrekt måste en del justeringar göras i den kod som innehåller inläsningsrutinerna (**problem 2.15**). Nedan listas de problem som hittats och som stämt överrens med denna kategoris karaktär:

PROBLEM #	BESKRIVNING
Problem 2.1	Filstrukturen är inte likadan i C++ och Java.
Problem 2.2	Tillgänglighet för variabler och metoder i klasser deklareras inte på samma sätt.
Problem 2.3	Arv deklareras i C++ med `:` och i Java med `extends`.
Problem 2.4	`unsigned` finns inte i Java.
Problem 2.5	Datatyperna i C++ och Java har inte samma storlek.
Problem 2.6	Höger-shift-operatorn `>>` i de båda språken fungerar inte på samma sätt.
Problem 2.7	Javas logiska shift-operator `>>>` fungerade inte så som beskrevs i litteratur.
Problem 2.8	Preprocessor-kommandot `#define` finns inte i Java.
Problem 2.9	Det reserverade ordet `const` som finns i C++ finns inte i Java.
Problem 2.10	Referenser finns inte i Java.

[246] Gosling, J. et al. *The Java Language Specification*, 2005.
[247] Webopedia.com, Online Computer Dictionary for Computer and Internet Terms and Definitions.
[248] Canadian Mind Products, *Java Glossary.*

Problem 2.11	Defaultparametrar finns inte i Java
Problem 2.12	Pekare finns inte i Java
Problem 2.13	Att nå ett visst specifikt ställe i en pekare fungerar inte på samma sätt som med Javas objekt.
Problem 2.14	Javas stränghantering och säkerhet
Problem 2.15	Little- och big-endian-systemet förändrar inläsning av filer

Tabell 14 Problematik rörande programspråkens skillnader

Diskussion och rekommendation kring problematik med programspråkens skillnader

Problem som berör skillnader mellan C++ och Java går så klart inte att undvika vid migrering. De flesta problem har enkla lösningar och på programkod som är enkel och strukturerad är det oftast inga problem att migrera rakt över med en del ändringar för syntaxspecifika och reserverade ord som skiljer sig åt i de olika programmeringsspråken. I den källkod som var objekt för denna studie återfanns inga klasser som kunde översättas direkt utan undersökning av funktionerna först.

Till problem som var enkla i detta experiment räknas till exempel **Problem 2.1** som behandlade att två filer slås ihop till en i Java. Detta löstes enkelt genom att de funktioner som ansågs viktiga för Javaapplikationen kontrollerades både i h och cpp-filen och slogs sedan samman till en funktion. **Problem 2.2** som behandlade tillgänglighetsgraden på variabler och metoder i klasser var inte heller något större problem, där skedde migrering rätt av med den förändringen att varje variabel och metod i Javakoden fick en tillgänglighets-variabel som prefix. Problemet med att C++ har två filer och Java samlar allt i en fil innebar även det att en extra noggrannhet fick tas då de två filerna i C++ skulle sättas ihop till en. Även arven, **problem 2.3**, gick enkelt att lösa genom att byta ut C++:s **:** mot Javas `extends`. Ett exempel på ett annat enkelt problem är **problem 2.4** som handlade om att variabler var deklarerade med `unsigned`. I vissa fall räckte det här att bara ta bort `unsigned` framför variabeldeklarationen. I andra fall krävdes dock att en variabeltyp med större omfång användes. Det var här viktigt att följa variabeln genom koden och kontrollera anledningen till att den är `unsigned` och sedan anpassa den nya koden efter det.

Problem 2.5 som hade att göra med datatypernas olika storlek gick att lösa enkelt. Så länge de variablerna inte användes tillsammans med bitoperatorer gavs Javaapplikationens variabler samma utrymme som variablerna i den ursprungliga applikationen krävde. Detta medförde att vissa variabler fick utökas genom användandet av en större variabeltyp. Problem med att `const` inte finns i Java, det vill säga **problem 2.9** orsakade inte något större problem i migreringen, oftast byttes `const` ut mot `final`. Problem med att defaultparametrar inte går att använda i C++, **Problem 2.11**, kan lösas genom att införa en ny metod för varje defaultparameter som finns i C++-koden. **Problem 2.12** och **problem 2.13** löstes genom att ta bort att pekare och ersätta dem med objekt. I de fall som pekare in i objektet krävdes, och detta var uteslutande strängmanipulering i den ursprungliga applikationen, kan `String`-arrayer användas i Java.

Vissa problem kräver lite mer krävande lösningar. **Problem 2.8** innebar att `#define` inte finns i Java, detta kan avhjälpas genom att en ny klass skapas där man definierar

klasskonstanter genom att ange prefixet `static` före Javas reserverade ord för konstanter: `final`. **Problem 2.10** med att Java inte stöder referenser kan undanröjas genom att skapa en `array` eller ett objekt av den variabel som behöver förändras även utanför den metod som den ingår som parameter i.

Problem 2.6 som ledde fram till **problem 2.7** var de problem som klassades som mest tidskrävande inom denna kategori. Dels var det svårt att hitta korrekt information i litteraturen. Sun Microsystems beskriver inte problemet i sin dokumentation över shift-operatorer och Javaspråket utan bortser från att det kan uppstå. Problemet innebar att migrera C++:s höger-shift-operation på en `unsigned char` till Java. I litteraturen står det att läsa att problemet kan avhjälpas med Java logiska höger-shift-operator (>>>) men detta råd fungerar inte då variabeln som ska shiftas är mindre än en `int`. Det står även att man kan migrera en `char` från C++ till Java utan att göra några ändringar och detta även om det är en `unsigned char` eftersom Javas `char` är en `byte` större, men detta fungerar inte då bitoperatorer ska användas eftersom bitarna då kommer förflyttas till fel plats. Som beskrevs tidigare användes i Javakoden `byte` istället för Javas `char`. Vid en höger-shift på fyra steg uppstod problem med att värdet som returnerades var felaktigt. Genom vidare studier i litteratur och förfrågningar på forum kunde problemet avhjälpas. För att få ett korrekt värde användes den logiska bitoperatorn `&`. Problem uppstod i Javakoden då denna kod användes: `this._byte[i] >>> 4` men då koden ersattes med `(this._byte[i] & 0xff) >>> 4` fungerade beräkningen korrekt. Detta beror på att hexadecimala konstanter som `0xff` ovan definierar en `int`, även om den beskriver en `byte`, och är därför det samma som `0x000000ff`.[249] Så genom att lägga till & `0xff` till koden ovan kommer den `int` som `byten` konverteras till enbart innehåller 0:or på de bitar som är större än själva `byten`. När talet då shiftas kommer 0:or att flyttas in i `byten` istället för 1:or, vilka hade gjort talet negativt.

Problem 2.14 som berörde att strängar inte kan raderas i Java löses genom användandet av `char`-arrayer vars värden går att manipulera. Little- och big-endian-problemet, **problem 2.15**, löses genom att programkoden för inläsningen skrivs om och den ordning i vilken `byte` ska läsas in anges explicit.

Generell kan sägas att det är viktigt att man har en bra bild av arkitekturen för den nya applikationen klar och då den befintliga applikationens arkitektur är uppbyggd på ett bra sätt är det lättast att använda sig av den. Det blir då lättare att kontrollera vad som migrerats och även lättare för utvecklarna av den ursprungliga applikationen att förstå den nya.

Det är också viktigt att kontrollera den befintliga källkoden, den speciella variabeln, klassen eller den speciella funktionen och se på dess användning, annars finns det risk för att migreringen blir felaktig. Det är viktigt att ta hänsyn till fall där programspråken har liknande syntax men resulterar i olika funktionalitet. I många fall kan det vara en bra idé att debugga till exempel en viss funktion i källkoden för att se hur den fungerar och vilka värden som laboreras med. Ibland kan det rent av vara tillfälle att bryta ut komplicerade funktioner från källkoden för att kunna testköra dem med olika parametrar och kontrollera flödet och dess utparametrar, härma beteendet för Javafunktionen och sedan kontrollera resultaten med varandra. Under experimentet har framkommit att då en funktion varit mycket komplicerad har det varit till nytta att ta ett steg tillbaka och se funktionen mer som en svart box. Att bortse

[249] Wagner, Niel, *The Laws of Cryptography: Unsigned bytes in Java*, 2001.

från hur C++-koden ser ut rent syntaxmässigt och istället se till den operation som utförs och sedan försöka efterlikna operationen och förändringen i Java.

En skillnad som är viktig att tänka på är fall där funktionaliteten är beroende av datatypers storlek eftersom dessa skiljer sig åt i programspråken. Även fall där programspråken skiljer sig från varandra i egenskaper som är av vikt för applikationen, såsom prestanda, minneshantering och säkerhet.

3: Problematik rörande datalagring

Denna kategori innefattar samtliga de problem som uppstått vid hanteringen av filer. Dessa filer lagrade dataapplikationens funktionalitet och var därför nödvändiga att bevara i någon form. Dessa hjälpfiler var av två slag. Den ena typen av filer var Windows register, som används för att lagra information om en PC i form av bland annat inställningar för hårdvara, mjukvara och användare. Detta register, vilket innefattar ett fåtal filer med lagrad data, är specifikt för Windows operativsystem (**problem 3.1**). En direkt motsvarighet till registret saknas i andra operativsystem varför det var nödvändigt att finna en annan möjlighet att lagra motsvarande uppgifter.

Den andra typen av data lagrades i filer av formatet XML (Extensible Markup Language). För att kunna presentera ett språkspecifikt utseende hos applikationen, lagrades ett antal textsträngar i dessa filer. Här var inte plattformsoberoendet något felande faktor eftersom filformatet kunde användas under ett flertal plattformar. Problemet med dessa utgjordes istället av dess lagringsplats. Filerna lagrades under för Windows-specifika kataloger (**problem 3.2**). Det var därför nödvändigt att finna en lagringsplats som kunde utnyttjas under samtliga plattformar.

Den lagringsplats som logiskt kunde ses var tillgänglig för dessa filer oavsett vilken plattform som användes, var programkatalogen under vilken applikationens källkod lagrades. Att kunna utnyttja denna katalog skulle förenkla hanteringen av samtliga de filer som applikationen var beroende av vid exekvering och skulle möjliggöra användandet av en och samma lagringsplats trots utnyttjandet av flertalet plattformar. Här uppstod nästa problem beroende av Windows restriktioner för lagring i just programkataloger. Windows har som standard att inte tillåta användaren att redigera i de kataloger där program är installerade (**problem 3.3**), vilket inte skulle ge möjlighet att använda denna typ av katalog för att lagra den data som applikationen krävde utan att ha möjlighet att förändra användarrättigheterna.

Ytterligare problem uppstod vid lagring av filer på grund av faktorer som behövde tas hänsyn till, i form av målplattformarnas olika användning av katalogseparatorer (**problem 3.4**). I senare versioner av Mac OS och i Linux används tecknet slash (/) som katalogseparator, medan tidigare versioner av Mac OS använder kolon (:) och Windows backslash (\). Dessutom har samtliga operativsystem olika system för filstrukturer, därför är även deras rötter specifika för respektive operativsystem(**problem 3.5**). Linux använder sig av slash (/) som rot, medan Windows rot definieras av C:. Macintosh filstruktur har en rot som kallas desktop-fil. Dock är den hierarkiska uppbyggnaden av filstrukturen gemensam för de olika operativsystemen. Man stöter på ytterligare skillnader mellan operativsystemen när man studerar informationen om en fil (**problem 3.6**) samt association mellan en fil och ett specifikt program (**problem 3.7**), vilket behandlas på olika sätt under Windows, Mac och Linux och som i vissa fall kan vara nödvändigt att ta hänsyn till.

PROBLEM #	BESKRIVNING
Problem 3.1	Windows register, vilket saknar motsvarighet i Linux och Mac OS, används för att lagra och hämta data berörande användare och system.
Problem 3.2	Hjälpfiler av formatet XML lagrades under för Windows specifika kataloger.
Problem 3.3	Windows har restriktioner för användarens rättigheter i programkataloger.
Problem 3.4	Operativsystemen utnyttjar olika tecken för katalogseparatorer.
Problem 3.5	Operativsystemen har olika filsystemsstruktur och rot.
Problem 3.6	Operativsystemen har olika sätt att identifiera filer.
Problem 3.7	Operativsystemen har olika sätt att associera filer och program

Tabell 15 Problematik rörande datalagring

Diskussion och rekommendation kring problematik med datalagring

Vid användning av plattformsspecifika hjälpfiler och hantering av sådana vid migrering, bör man antingen konvertera filerna till ett specifikt format för målplattformen, eller använda sig av ett format som kan utnyttjas av flera plattformar. Det rekommenderas allmänt vid migrering att man dessutom strävar så långt möjligt efter att nå samma bas för båda applikationerna, vilket gäller både struktur, källkod och filer.

Den ursprungliga applikationen utnyttjade, som ovan beskrivits, filer av formatet XML för lagring av en typ av textsträngar. På grund av formatets portabilitet var det önskvärt att behålla dessa filer intakta, vilket också krävde att funktionalitet för att hantera dessa var nödvändiga att migrera. Därför beslutades att även lagra andra nödvändiga uppgifter i filer av detta format. Detta innebar även att det var nödvändigt att hantera skillnader mellan hanteringen av filinformation i samband med associationen mellan filerna och rätt program för att visa dem. Mer om rekommendationer för lösning av detta problem finns under problemkategorin som berör externa bibliotek och komponenter. För att undvika att filerna under något operativsystem skulle vara otillgängliga följdes de rekommendationer rörande benämning av filer som följer:

RESTRIKTION	REKOMMENDATION
Filassociation	Filnamnet ska följas av lämplig filändelse.
Filnamnens längd	Längden begränsas till den lägsta av operativsystemens högsta tillåtna filnamnslängd.
Otillåtna tecken	Filnamnen utnyttjar endast vanliga tecken, såsom bokstäver A-Z samt siffrorna 0-9.
Skiftlägeskänslighet	Konsekvens av användandet av versaler och gemener i namngivningen.

Tabell 16 Filrekommendationer

Då migreringen bestod av att stödja exekvering på ett flertal plattformar var det nödvändigt att hantera ett motsvarande antal filsystemsstrukturer, närmare bestämt i form av operativsystemens olika rot och katalogseparatorer. I det här fallet, då Java utnyttjades som utvecklingsspråk, var problemet av mindre karaktär då Javas egenskap av att ge möjlighet att utveckla plattformsoberoende applikationen gav stöd i hanteringen. Java gör dessa systemspecifika uppgifter tillgängliga genom metodanrop som returnerar rätt form av separatorer beroende av vilket operativsystem som applikationen exekveras på.[250] [251] [252]

4: Problematik rörande externa bibliotek och komponenter

I denna kategori har samlats de problem som rör externa bibliotek och komponenter. Det rör sig dels om problem kring de bibliotek och komponenter som den befintliga applikationen använde sig av, men också deras motsvarighet i Javavärlden och problem kring dem.

De egenskaper som problem i denna kategori har är att de kallas på ifrån applikationens källkod och kan vara komponenter bestående av mer av applikationens källkod, alternativt kod från ett helt externt bibliotek. Applikationens källkod skickar med data till de externa biblioteken eller komponenterna och dessa utför det som källkoden ber dem att utföra. Ibland innebär det att fristående externa applikationer hanterar data som skickats med utan någon vidare interaktion med källkoden. Ibland innebär det att externa bibliotek hanterar medskickad data, förändrar den eller returnerar ett värde som förändrats utifrån inskickat data.

De problem som innefattas i denna kategori är av en övergripande karaktär och ger sig inte i kast med kodspecifika problem. Problemen har ofta krävt en djupare utredning innan de kunnat förstås eller lösas. Vidare har problemen bara inträffat eller upptäckts en gång var under experimentet, det är alltså inte återkommande problem utan kräver bara en lösning.

Venners skriver om Dynamic Link Library-filer (dll-filer) att de är plattformsspecifika, men att de ändå kan laddas in i Java till exempel genom att använda JNI.[253] För att dll-filerna ska gå att använda i en plattformsoberoende applikation måste JNI användas och programkoden förändras för att passa JNI.[254] Ett problem som var något av det första som stöttes på vid genomgång av koden var att kärnan i källkoden anropade moduler paketerade som dll-filer. Eftersom en plattformsoberoende Javaapplikation anropar dll-filer genom JNI gick inte anropen i den befintliga applikationens programkod att använda. dll-filerna innehöll plattformsspecifik kod och var kompilerade för Windows-plattformen och de gick därför inte att använda i ett program som ska vara plattformsoberoende (**problem 4.1**).

Ett annat problem är de filer som Java-kompilatorn skapar, nämligen class-filerna. Dessa filer är kompilerade till bytekod, men är strukturellt uppbyggda och därför relativt enkla att göra reverse engineering på.[255] Detta innebär att den programkod som skrivits av programmeraren går att utvinna ur class-filen (**problem 4.2**). Eftersom den programvara som experimentet utfördes på är kommersiell och ska hantera säkerhet är det av yttersta vikt att programkoden inte kan läsas av andra än de som den är ämnad för.

---

[250] Hall, Marty och Brown, Larry, *"Core web programming – Java Input/Output"* , 2003.
[251] Sun Microsystems, "*Java 2 Platform Standard Edition 5.0 Api*", 2004.
[252] Khan, Asim, "*Java jukebox – Java input & output*".
[253] Venners, Bill, *Inside the Java Virtual Machine*.
[254] Campione, Mary et al, *The Java Tutorial.*
[255] Harold, Elliotte Rusty, Hemligheterna i Java. (Jönköping: IDG AB, 1997).

Som nämndes i teorin finns det många olika webbläsare, där den mest populära är Internet Explorer. Internet Explorer finns tillgänglig för Windows och upp till version 5 finns webbläsaren även tillgänglig för Mac OS, däremot finns den inte att tillgå för Linuxanvändare.[256] Den befintliga applikationen använde sig av webbläsaren Internet Explorer för att visa upp dekrypterade filer (**problem 4.3**). Då webbläsaren inte finns tillgänglig för alla operativsystem som är väsentliga för detta experiment uppstod problem. Ett annat problem som uppstod och som är relaterat till webbläsare är att alla webbläsare inte har exakt likadan funktionalitet inbyggd och därför kunde inte en bra presentation av den dekrypterade filen garanteras (**problem 4.4**).[257]

För att hantera säkerheten i den befintliga applikationen används bland annat ett säkerhetsbibliotek vid namn Crypto++. Detta bibliotek finns tillgängligt för Windows, Mac OS och Linux, men bara för programkod som är skriven i C++, det finns alltså inte för Java (**problem 4.5**).[258] Till Java finns det alternativ som till exempel Javas egna säkerhetsbibliotek Java Cryptography Extension (JCE) och Bouncy Castle men metodnamn och funktioner motsvarar inte Crypto++:s funktioner (**problem 4.6 och 4.7**).[259,260] Då programmet ska dekryptera filer som krypterats med Crypto++ måste dekrypteringen ske på exakt samma sätt som det gör i den befintliga applikationen, annars kommer innehållet inte att kunna visas.

Den befintliga applikationen använder sig av ett bibliotek som heter Synchronous Key Generator (SKG). SKG finns genom ett JNI-interface tillgänglig för Java men detta gör att Javaapplikationens plattformsberoende styrs till att bara innefatta just de operativsystem som SKG:n stödjer och gör därigenom Javaapplikationen mer plattformsberoende (**problem 4.8**).[261] I dagsläget stödjer SKG:n operativsystemen Windows, Mac OS, Linux och Unix.[262]

Ett problem som är mer applikationsspecifikt är att all C++-kod som interagerar med SKG:n i den befintliga applikationen måste skrivas om (**problem 4.9**). I C++ sker i princip all interaktion med SKG:n med anrop genom objekt. I Javaversion erbjuder SKG:n inget sätt att skapa objekt utan man får arbeta med siffror för att komma åt den data som krävs. C++-koden arbetar konstant med att skicka parameterdata som förändras, eftersom detta inte är möjligt i Java måste alla de variabler som skickas göras om till objekt och arrayer.

Den grafiska presentationen av applikationen i experimentet var i huvudsak byggt med hjälp av Microsoft Foundation Classes (MFC). Biblioteket är en samling C++klasser som formar ett ramverk för applikationer med vilket man kan hantera objekt och fördefinierade fönster och vanliga grafiska komponenter.[263] Problemet som uppstod här bestod i att MFC var språkspecifikt för C++ (**problem 4.10**).

[256] Microsoft, *Internet Explorer Home.*
[257] Knight, D et al, *Platform Independence – Final report*, 2001.
[258] Eskimo, *Crypto++ Library 5.2.1*
[259] Sun Microsystems, *Java Cryptography Extension (JCE).*
[260] Bouncy Castle, *The Legion of the Bouncy Castle.*
[261] Venners, Bill, *Inside the Java Virtual Machine*.
[262] Impsys Digital Security AB, *The Home of Digital Security*.
[263] Wikipedia, The Free Encyclopedia, *Microsoft Foundation Classes.*

PROBLEM #	BESKRIVNING
Problem 4.1	Källkoden har gjorts modulär genom användandet av dll-filer vilka inte är plattformsoberoende.
Problem 4.2	Java kompileras till class-filer som i sin tur lätt kan dekompileras och på så sätt kan källkoden kommas åt.
Problem 4.3	Applikationen hade en plattformsspecifik lösning som inkluderade webbläsaren Internet Explorer för uppvisning av dekrypterade filer.
Problem 4.4	Olika webbläsare har inte samma funktionalitet.
Problem 4.5	Källkoden använder sig av säkerhetsbiblioteket Crypto++ vilket endast finns tillgängligt för C++.
Problem 4.6	Javaapplikationen använde sig av JCE vilket inte hade motsvarande funktioner som Crypto++.
Problem 4.7	Javaapplikationen använde sig även av säkerhetsbiblioteket Bouncy Castle vilket inte hade motsvarande funktioner som Crypto++.
Problem 4.8	Applikationen använder sig av en extern modul kallad SKG. SKG gjorde att Javaapplikationen blev mer plattformsberoende.
Problem 4.9	Interaktionen mellan SKG och Java blir kraftigt förändrad jämfört med samma interaktion i C++-koden.
Problem 4.10	Applikationens grafiska gränssnitt använder sig av biblioteket MFC vilket är plattformsspecifikt.

Tabell 17 Problematik rörande externa bibliotek och komponenter

Diskussion och rekommendation kring problematik med externa bibliotek och komponenter

Problem 4.1 hade att göra med att den undersökta applikationen laddade in plattformsspecifika dll-filer. Detta löstes genom att varje dll-fil undersöktes och de viktiga funktionerna lokaliserades. Detta gjordes genom debug men även genom förfrågningar till programmerarna som varit med och byggt den undersökta applikationen. Alla de funktioner som klassificerades som betydelsefulla för Javaapplikationen fick sedan migreras till Javaspråket och plattformsoberoende.

class-fils-problemet, **problem 4.2**, finns det ingen riktigt bra lösning på. Programmerare i forum och Internetsidor ger samma svar, det finns inget som kan hindra någon från att komma åt källkoden i filerna, frågan är bara hur komplext det ska vara. Att dekompilera en class-fil som inte skyddats är inte några problem, och det finns gott om verktyg som gör detta åt den nyfikna. För att skydda sig något mot problemet, och för att bjuda motstånd kan man använda sig av något som kallas för Obfuscator. Detta är ett verktyg som mekaniskt ersätter alla metoder- och variabelnamn med slumpmässiga och meningslösa namn. Obfuscating kan göras mer eller mindre komplex, där de mest komplexa lösningarna oftast ägs av företag och blir en ekonomisk fråga. Det mest extrema sättet att skydda sig mot reverse engineering av class-filerna är att skriva sin egen class-loader. Ett annat alternativ är att kryptera filerna då de inte

används och dekryptera dem då de ska användas men då krävs en nyckel för dekrypteringen och den måste ju sparas så att programmet kommer åt den.

Problem 4.3 bestod i att den undersöka applikationen var direkt kopplad till webbläsaren Internet Explorer. Detta problem kan lösas genom att i Javaapplikationens inställningshantering lägga till ett väl där användaren själv kan specificera den webbläsare som denna vill använda. På så sätt görs applikationen också mer användarorienterad då användaren inte tvingas till att använda en specifik webbläsare. För att lösa **problem 4.4** och därmed få en försäkran om att presentationen av dekrypterade filer visas på ett representativt sätt kan rekommendationer läggas in om vilka webbläsare som är lämpliga att använda, men användaren kan ändå ha möjlighet att välja ett annat alternativ. Detta ger också möjlighet att hantera visningen av filerna utan att behöva hantera filassociation eftersom information om vilken webbläsare som ska användas vid filöppnandet finns att tillgå. Med hjälp av denna information kan Javametoder nyttjas för att visa en fil i angiven webbläsare.

Problemen kring säkerhetsbiblioteken, **problem 4.5**, **problem 4.6**, **problem 4.7** tog mycket tid i anspråk. Dels tog det tid att kontrollera hur speciella Crypto++-funktioner fungerade och sedan att försöka hitta säkerhetsbibliotek med exakt likadana funktioner. Många hjälpfunktioner fanns inte i de andra biblioteken och vissa säkerhetskritiska funktioner fungerade inte på exakt samma sätt i Javas utbud av säkerhetsbibliotek eller det externa biblioteket Bouncy Castle.

Problem 4.8 som hade att göra med användandet av SKG och JNI-interfacet löstes genom att hela interaktionskoden mellan den undersökta applikationen och SKG ersattes med ny kod. JNI fungerar ju så att programmet som använder sig av JNI skickar med olika uppsättningar av bibliotek, i detta fall dll och so. Då programmet körs laddas det bibliotek som är lämpligt för operativsystemet in i applikationen. Det som är viktigt att tänka på här är att inte glömma att distribuera biblioteken tillsammans med applikationen och att tänka på att det är rätt versioner av biblioteken som distribueras. **Problem 4.9** berörde också SKG-komponenten och hade att göra med att interaktionen mellan applikationen och komponenten blev kraftigt förändrad i Javaapplikationen jämfört med den ursprungliga applikationen. Detta problem löses genom en total omskrivning av koden.

Problem 4.10 utgjordes av att det grafiska gränssnittet var plattformsspecifikt. Det finns två möjligheter att skapa grafiska användargränssnitt som ska användas på ett flertal plattformar. Det ena är att generera en uppsättning kod med plattformsspecifika komponenter för varje plattform som applikationen är tänkt att användas på. Denna lösning är dock tidskrävande och innebär att applikationen i sin helhet inte kommer att vara helt plattformsoberoende. Den andra möjligheten är att använda sig av verktyg för grafiska gränssnitt som genererar plattformsoberoende komponenter. Det rekommenderas att man undviker att specificera komponenters faktiska placering och istället använder sig av layoutmanagers. Man bör inte heller anta en viss skärmstorlek eller upplösning, eller tillgänglighet av olika typsnitt.

Då C och C++ är programspråk som används av ett flertal programmerare är också en mängd verktyg för gränssnitt designade för att använda dessa språk, såsom QT, Fox, wxWidgets, Openstep och GTK. Eftersom en objektorienterad lösning är applicerbart på grafiska gränssnitt, använder sig de flesta av dem av C++. Det finns dock ett antal gränssnittsverktyg som använder sig av andra programspråk eller som är språkoberoende, såsom Fresco. Fresco

använder sig av IDL (Interface Definition Language) med vilket man kan specificera gränssnitt för objekt, vilket har egenskapen av att vara språkoberoende.[264]

Val som gjordes resulterade i ett behov av ett gränssnitt som oberoende av plattform kunde exekveras med samma funktionalitet, och med en kodbas. I ett tidigt skede hade avgjorts att Java skulle användas för utveckling av koden för kärnan, därför föll det sig naturligt att studera Javas möjligheter att uppfylla de krav som ställdes på gränssnittets portabilitet och utseende. Om man anser det nödvändigt att ange ett specifikt utseende för en viss plattform ger Java möjlighet att ta reda på vilken plattform programmet exekveras på och ändra inställningarna därefter. I annat fall finns möjlighet att utnyttja Javas Swing-pakets funktionalitet för att visa ett anpassat gränssnittsutseende utifrån den plattform som används.

Här följer några generella rekommendationer inom hanteringen av externa bibliotek och komponenter för att uppnå plattformsoberoende. I det fall det inte går att använda de bibliotek och komponenter som originalapplikationen använder är det viktigt att försöka hitta motsvarande som är så lika originalapplikationens externa bibliotek och komponenter som möjligt. Precis som alltid när det gäller programmering för plattformsoberoende gäller det att hitta den minsta gemensamma nämnaren för plattformarna. I de fall där det är svårt att hitta en komponent som passar alla operativsystem är det, i de fall det är möjligt, lämpligt att låta användaren specificera komponenten som ska användas. Vissa applikationer är beroende av att exakt likadana komponenter och bibliotek används. I det fallet är det viktigt att kontrollera dessa hur de fungerar ihop med andra plattformar och det programmeringsspråk som är tänkt att användas innan vidare utredning görs. Om komponenten eller biblioteket bara finns för en viss plattform så blir denna plattform applikationens minsta gemensamma nämnare och en migrering för att uppnå plattformsoberoende kan bli omöjlig att genomföra utan stora förändringar i och nyskrivande av program kod för den nya applikationen eller utan att migrera den plattformsspecifika komponenten.

[264] Babcock, Michael, *The Importance of the GUI in Cross Platform Development*, 1998.

Generell diskussion

Med detta avsnitt har vi för avsikt att visa områden som är av intresse för fortsatt forskning, samt påvisa bredden av, och validiteten i, resultatet som här framkommit. Avsikten är att diskutera problemen ur mer generell aspekt för att göra det möjligt att applicera rekommenderade lösningar på migrering i allmänhet snarare än för denna för uppsatsen specifika process. Migreringsprocessen i experimentet diskuteras vidare, dels för att påvisa problematikens uppkomst, dels för att påvisa alternativa lösningsförslag. Avsnittet avslutas med sammanfattande rekommendationer vid migrering.

Att migrera källkod är på inget sätt en ny företeelse. Det finns en mängd information om migrering mellan olika plattformar att finna. Utvecklare av operativsystem delar mer än gärna med sig av information som berör hur man migrerar applikationer för att kunna exekvera dem på deras plattform. Av förklarliga skäl är de mer restriktiva med information som ger stöd vid migrering för plattformsoberoende. Då vi sökt efter tidigare forskning inom området har mycket få artiklar hittats som behandlar mer än endast delar av det. Främst behandlade artiklarna forskning kring Javas virtuella maskin och dess plattformsoberoende egenskaper, forskning kring plattformsoberoende GUI-hantering och utveckling samt webbutveckling. Vi har istället använt oss av böcker, forum och whitepapers som vi bedömt vara tillförlitliga. Vidare undersökningar och forskning är därför intressant inom detta område, det kan vara en fördjupning av en liknande undersökning som vår, men det är även intressant att undersöka problem som berör andra plattformar eller involverar andra programspråk.

Syftet med uppsatsen: att kartlägga och beskriva problem som kan uppstå då en Windows-applikation migreras för att göras plattformsoberoende anser vi ha uppnått. De problem som påträffats under det tre månader långa experimentet är inte heltäckande, men de påvisar förekommande migreringsproblem och även komplexiteten som finns kring migrering för att uppnå plattformsoberoende.

Experimentet sågs till en början som en relativt enkel uppgift då det avsåg migrering av endast en del av den fullständiga programvaran. Att applikationen i huvudsak bestod av, och skulle komma att bestå av, kod skriven i av för oss kända programspråk gjorde också att uppgiften uppfattades vara av omfattning och svårighetsgrad passande experimentets och uppsatsens begränsade tidsram. Snart fick vi dock erfara att en vid första anblicken enkel uppgift lätt kunde försvåras av en mängd problem som uppstod under experimentets gång.

Den ursprungliga applikationens egenskaper gjorde att migreringen resulterade i problem av flera slag. Avsaknad av dokumentation, applikationens beroende av ett flertal externa bibliotek samt det faktum att applikationen inte i något avseende var utvecklad för plattformsoberoende gjorde att problemen som uppstod givit uppsatsen resultat av signifikant bredd.

De allmänna riktlinjer som finns för migrering rekommenderar att man i startfasen lägger stor vikt vid att förstå den ursprungliga applikationen och dess funktioner. Avsaknaden av dokumentation gjorde att ett försök till reverse engineering utfördes för att få en översikt av applikationen. Denna process gav inte det önskade resultatet på grund av att applikationen gjorde anrop till dll-filer och därför inte kunde presenteras grafiskt i sin helhet. Detta gav till följd att en helt manuell genomgång av en omfattande mängd kod fick göras, med förhoppningen att reda ut vilka rader av kod som var av betydelse för den del av applikationen som skulle migreras, hur arkitekturen var byggd, hur modulerna samverkade, samt vilka

beroenden som fanns mellan källkodskärnan, grafiska gränssnitt samt externa bibliotek och komponenter. Den här processen var så pass tidskrävande att förstudien i experimentet blev betydligt mer utdragen än vad som förväntats. Om förutsättningarna varit annorlunda, till exempel genom att vi varit förtrogna med källkoden sedan tidigare eller genom att dokumentation fanns tillgänglig över applikationen skulle denna fas och därmed migreringen inte ha tagit så mycket tid i anspråk.

En mindre omfattande studie av konverteringsverktyg för programspråk resulterade i vetskapen om att dessa var ett bra stöd vid omvandling av kod i de fall koden var skriven uteslutande i det programspråk från vilket man skulle översätta. Utifrån det faktum att applikationen utnyttjade en stor mängd externa bibliotek och komponenter gjordes bedömningen att ett verktyg för att konvertera kod skriven i ren C++ till Java-kod i jämförelse med en manuell omvandling inte skulle vara nämnvärt tidsbesparande varför detta alternativ lämnades åt sidan. I experimentet användes nära besläktade programspråk men ändå uppstod en stor mängd problem vid migreringen inom denna kategori. Om den befintliga applikationens källkod varit skriven i ren C++ och inte använt sig av externa bibliotek och komponenter skulle ett konverteringsverktyg vara en bra lösning på kodmigreringsproblemet. Genom användning av ett sådant verktyg skulle mycket tid kunna sparas.
Då beslut tagits om migreringsstrategi fortsatte vi vårt experiment med att lokalisera de funktioner i den befintliga applikationens källkod som ansågs viktigast för att få den plattformsoberoende applikationen körbar så snart som möjligt. Det första som skapades var applikationens gränssnitt. Detta gjordes så enkelt som möjligt med Javas Swingbibliotek, vilket vi anser vara det bästa alternativet för ett plattformsoberoende användargränssnitt.

För att lokalisera andra elementära funktioner använde vi oss av debugging, men eftersom applikationen var mycket stor, komplext uppbyggd och vi hade för lite insikt i hur den fungerade blev detta ett mycket tidsödande arbete. I detta skede fick vi hjälp av dem som varit med och konstruerat den befintliga applikationen. Då ingången till de viktiga funktionerna hittats kunde resten av funktionerna enkelt hittas genom ytterligare debugging. Funktionerna som påträffades och som kategoriserades som viktiga använde sig av diverse externa bibliotek vilka var mycket svåra att hitta en passade motsvarighet till. Då vårt experiment skulle utföras inom en begränsad tid koncentrerades vi vår lösning på att hitta motsvarande funktionalitet i inbyggda såväl som externa Javabibliotek.

Då den befintliga applikationen använde sig av bibliotek med öppen källkod som dessutom fanns kompilerade för Windows, Mac OS och Linux skulle problemet mycket väl ha kunnat avhjälpas genom användning av de viktiga funktionerna i biblioteken som den befintliga applikationen nyttjade. Detta skulle medföra att JNI skulle användas som gränssnitt mellan biblioteksfunktionerna och Javaapplikationen. Anledningen till att vi inte tittade närmre på detta förslag var som tidigare nämnts tidsbrist, men framförallt att applikationen då skulle bli beroende av två JNI-lösningar, vilket komplicerar distribution och underhåll av applikationen.

Ytterligare en lösning på migreringen i fråga hade varit att generera kravspecifikationer och designdokument grundade på de funktioner i den ursprungliga applikationen som var nödvändiga att bevara och utifrån dessa dokument utveckla en applikation i det nya programspråket, helt fristående från den ursprungliga. I det här fallet hade den lösningen förmodligen varit fördelaktig på grund av avsaknaden av dokumentation för den ursprungliga källkoden, vilket gjorde koden både svår att förstå, förändra och underhålla. Det hade medfört ett arbete med att skapa funktionalitet snarare än att finna direkta motsvarigheter till funktionernas ursprung. Möjligen hade detta synsätt givit större möjlighet att utnyttja

programspråk och bibliotek på ett bättre sätt. Att lyfta blicken från detaljerna och istället se till helheten hade troligen resulterat i, inte bara utvecklingsfrihet, utan även kvalitetsförbättringar för applikationen som helhet.

Denna uppsats har påvisat att de initiala faserna av migreringsprocessen är av stor betydelse för resultatet. Framgången med migrering är beroende av utvecklarnas kunskap om såväl den ursprungliga applikationen som programspråk, bibliotek, verktyg och plattformar. En bred, allmän kunskap om dessa delar är fördelaktig vid valet av verktyg, utvecklingsmiljö och tillvägagångssätt, medan en djup kunskap om de som valts för migreringen är nödvändig för att nå önskat resultat. Att utvecklaren själv erhåller denna bredd av kunskap kan även med tidsbesparande fördelar delvis ersättas med, eller kompletteras av, kunskapskällor i form av dokumentation, andra utvecklare, litteratur eller annan typ av resurser. Det inledande stadiet bör resultera i tydliga riktlinjer för det fortsatta arbetet, men bör också ge möjlighet att återgå till tidigare stadier i de fall man stöter på problem. Uppsatsens resultat har också påvisat att det för problemen som uppstått vid flera tillfällen funnits ett flertal lösningar. Det är därför fördelaktigt att speciellt under startfasen, men även under pågående migreringsprocess, lyfta blicken från detaljerna, se till helheten och om möjligt söka lösningar att angripa problemet ur en annan vinkel. En bredd av lösningsförslag och kunskap om dessa resulterat troligen i ett kvalitetsmässigt bättre resultat.

Avslutningsvis kan sägas att migrering för att uppnå plattformsoberoende är en process vars komplexitet är beroende av ett stort antal faktorer. Även om verktyg finns att tillgå för att konvertera programspråk mellan varandra är det inte alltid dessa kan användas. Även om ett programspråk med vilket man har möjlighet att skapa plattformsoberoende källkod används, kommer problematik kring de olika plattformarna ändå att uppstå. Den migreringsuppgift som vid första anblicken ser lätt ut kan visa sig vara mycket komplicerad.

Referenslista

About.com, *Focus on Linux*, 2005, [WWW-dokument] URL http://linux.about.com/od/linux101/l/blnewbie3_1_1.htm Access 2005-04-30

Answers.com, *Bitwise operations* [WWW-dokument] URL http://www.answers.com/topic/bitwise-operation Access 2005-04-16

Babcock, Michael, *The Importance of the GUI in Cross Platform Development*, 1998, [WWW-dokument] URL http://www2.linuxjournal.com/article/2723 Access: 2005-04-12

Backman, Jarl, *Rapporter och uppsatser,* Lund, Studentlitteratur, 1998

Best, Jo, *Survey: Some iPod fans dump PCs for Macs*. [WWW-dokument] URL http://news.com.com/Survey+Some+iPod+fans+dump+PCs+for+Macs/2100-1042_3-5465935.html?tag=st.rc.targ_mb CNET News.com 2004. Access 2005-04-02

Bohlin, Stefan, *Alla vill sälja Ipod och Mac*. *Computer Sweden* Nr 40 (2005): 9

Bouncy Castle, *The Legion of the Bouncy Castle* [WWW-dokument] URL http://www.bouncycastle.org/ Access 2005-04-28

Browser News, *Resources - Fonts*, 2005 [WWW-dokument] URL http://www.upsdell.com/BrowserNews/res_fonts.htm Access 2005-05-19

Cadenhead, Rogers och Laura Lemay, *Sams Lär dig Java 2 på 3 veckor*. Jim Wibom. Upplaga 3. Falun: Pagina Förlags AB, 2003

Campione, Mary och Kathy Walrath och Alison Huml, *The Java Tutorial*. [WWW-dokument] URL http://java.sun.com/docs/books/tutorial/index.html Access 2005-04-09

Campione, Mary och Kathy Walrath. *About the Java Technology*. [WWW-dokument] URL http://java.sun.com/docs/books/tutorial/getStarted/intro/definition.html Access 2005-04-02

Canadian Mind Products, *Java Glossary* [WWW-dokument] URL http://mindprod.com/jgloss/endian.html Access 2005-05-03

Chapman, Lindell, *File systems*, 2004 [WWW-dokument] URL http://faculty.nhmccd.edu/ochapman/docs_used/itsc1405/ch_03ppt.doc Access 2005-04-27

Chopra, Manish, *Cross-Platform Computing*, 2002 [WWW-dokument] URL http://www.techtherightway.net/software/sun/html/cpcsem4.htm Access 2005-04-27

Codemartial, *The Structure of Linux*, 2005 [WWW-dokument] URL http://www.codemartial.org/Linux/ii.shtml Access 2005-04-27

Comentum, *Comparison of File and Directory Name and File Systems on Macintosh, PC and Unix Computers and Operating systems* [WWW-dokument] URL http://www.comentum.com/File-Systems-HFS-FAT-UFS.html Access 2005-04-18

Computer Hope, *Case Sensitive* [WWW-dokument] URL http://www.computerhope.com/jargon/c/casesens.htm Access 2005-04-18

Cooperman, Gene, *Converting Java to C++* [WWW-dokument] URL http://www.ccs.neu.edu/course/csu215/review/java-to-c++.html Access 2005-04-07

Coustan, Dave och Curt Franklin, *How Operating Systems Work* [WWW-dokument] URL http://computer.howstuffworks.com/operating-system.htm HowStuffWorks, Inc. Access 2005-04-04

Cplusplus.com, *Description of C++* [WWW-dokument] URL http://www.cplusplus.com/info/description.html Access 2005-05-15

Cplusplus.com, *History of C++* [WWW-dokument] URL http://www.cplusplus.com/info/history.html Access 2005-05-15

Cravotta, Robert, *Fear and loathing of porting embedded software* [WWW-dokument]. URL http://www.edn.com/article/CA236414.html Access 2005-04-13

Crawford, Michael D, *Writing Cross-Platform Software – Getting Started* [WWW-dokument] URL http://www.goingware.com/tips/getting-started/. 2002 Access 2005-04-02.

Dartmouth, *Storing information – The file system,* 2004 [WWW-dokument] URL http://www.dartmouth.edu/~rc/classes/unix1/slide13.frameset.html Access 2005-04-27

Davis, Mark, *Porting C++ to Java* [WWW-dokument] URL http://www.tunzi.ch/Java_Cookbook.pdf, 1997 Access 2005-04-02

Developer Connection, *BSD General Commands Manual* [WWW-dokument] URL http://developer.apple.com/documentation/Darwin/Reference/ManPages/man1/defaults.1.html Access 2005-04-27

Developer.com, *UML Tools* [WWW-dokument] URL http://www.developer.com/design/article.php/1593811 Access 2005-04-14

Doughty, Mike, *Understanding Macintosh and Windows File Types*, 2003 [WWW-dokument] URL http://www.sketchpad.net/filetypes.htm Access 2005-04-27

Eckel, Bruce, *Comparing C++ and Java*, 2000 [WWW-dokument] URL http://www.javacoffeebreak.com/articles/thinkinginjava/comparingc++andjava.html Access 2005-04-28

Eckel, Bruce, *Thinking in Java*. Prentice-Hall; 3:e upplagan, 2002

Eskimo, *Crypto++ Library 5.2.1* [WWW-dokument] URL http://www.eskimo.com/~weidai/cryptlib.html Access 2005-04-16

Express Design Inc, *Unixguide – File System Layout* [WWW-dokument] URL http://www.ed.com/unixguide/fs.html Access 2005-05-02

Fleming, R. Mark, *Cross-platform compatible files – File Naming Issues,* 2002 [WWW-dokument] URL http://www.imagemontage.com/Docs/FileNames.html Access 2005-04-27

Frechtling, Joy och Laure Sharp, *User-Friendly Handbook for Mixed Method Evaluations* [WWW-dokument] URL http://www.nsf.gov/pubs/1997/nsf97153/start.htm National Science Foundation, 1997 Access 2005-04-02

Genzuk, M., *A synthesis of ethnographic research*. Southern California: University of Southern California Center for Multilingual, Multicultural Research

Gosling James, Bill Joy, Guy Steele och Gilad Bracha, *The Java Language Specification*. Addison-Wesley; 3:e upplagan, 2005

Gosling. James, *The Java language environment* [WWW-dokument] URL http://java.sun.com/docs/white/langenv/, 1996 Access 2005-04-02

Graham, Alan, *Home on the Go with NetInfo* [WWW-dokument] URL http://www.macdevcenter.com/pub/a/mac/2002/02/08/netinfo.html Access 2005-04-27

Gunnarsson, Ronny, *Forskningsansats – kvalitativt eller kvantitativt perspektiv* [WWW-dokument] URL http://www.infovoice.se/fou/bok/kvalmet/10000002.htm 2002. Access 2005-04-16

Gunnarsson, Ronny, *Grounded theory* [WWW-dokument] URL http://www.infovoice.se/fou/bok/kvalmet/10000011.htm 2002. Access 2005-04-11

Hall, Marty och Brown, Larry, *Core web programming – Java Input/Output* , 2003 [WWW-dokument] URL http://notes.corewebprogramming.com/student/Java-IO.pdf Access 2005-05-02

Harold, Elliotte Rusty, *Hemligheterna i Java*. Jönköping: IDG AB, 1997

Hancock, Beverley, *An Introduction to Qualitative Research*. University of Nottingham, 1998

Hokum Glossaries, *File extensions*, 2004 [WWW-dokument] URL http://hokum.freehomepage.com/Content/Glossary/Glossary_FileEx.html Access 2005-04-27

Hughes, J., *Moving Out from the Control Room: Ethnography in System Design*. Lancaster: Lancaster University,1994

Imagemontage, *Cross-Platform Compatible Files*, 2002 [WWW-dokument] URL http://www.imagemontage.com/Docs/DOSSuffix.html Access 2005-04-27

Impsys Digital Security AB, *The Home of Digital Security* [WWW-dokument] URL http://www.impsys.se/ Access 2005-05-11

Kalev, Danny, *Porting a C++ Application to Java*, 1998 [WWW-dokument] URL http://www.stat.sinica.edu.tw/library/cd_database/C%20C++users%20Journal/html/16.02/kalev/kalev.htm Access 2005-04-11

Kelley, Allan, *Where to begin: Porting,* 2001 [WWW-dokument] URL http://www.portinggurus.com/Navigator.asp?link=http://www.arnodevelopment.co.uk/writing/Overload/Porting/Porting1.pdf Access 2005-04-13

Khan, Asim, "*Java jukebox – Java input & output*" [WWW-dokument] URL http://www.geocities.com/javajbox/io.htm Access 2005-05-02

Knight, David, Thurley, James och Wittams, Rob, *Platform Independence – Final report*, 2001 [WWW-dokument] URL http://infoeng.ee.ic.ac.uk/~malikz/surprise2001/jbt99e/report/ Access 2005-04-29

Lewis & Clark College, *Getting started with OS X* [WWW-dokument] URL http://www.lclark.edu/~infotech/HELP/HELPSHEETS/gettingstartedosx.pdf Access 2005-04-20

Lindholm, Tim och Frank Yellin, *The JavaTM Virtual Machine Specification*. Addison-Wesley Professional, 2:a upplagan, 1999

Linux Forum, *General overview of the Linux file system*, 2003 [WWW-dokument] URL http://www.linuxforum.com/linux/sect_03_01.html Access 2005-05-13

Litvin, Maria and Litvin, Gary, *Java Methods,* Skylight Publishing, 2001 [WWW-dokument] URL http://www.skylit.com/javamethods/appxa.pdf Access 2005-04-12

Martin, Johannes, *Ephedra A C to Java Migration Environment,* 1996 [WWW-dokument] URL http://www.notamusica.de/home/jmartin/resume/dissertation.pdf Access 2005-04-12

Merriam, B Sharan, *Fallstudien som forskningsmetod*, Björn Nilsson. Lund: Studentlitteratur, 1994

Microsoft, *From One Code Base to Many Platforms Using Visual C++* [WWW-dokument] URL http://msdn.microsoft.com/archive/default.asp?url=/archive/en-us/dnarvc/html/msdn_fromone.asp Access 2005-04-18

Microsoft, *Internet Explorer Home* [WWW-dokument] URL http://www.microsoft.com/windows/ie/default.mspx Access 2005-04-28

Mindfire Solutions, *Mac OS-Windows Porting* [WWW-dokument] URL http://www.mindfiresolutions.com/download/Porting=Mac-Win%20Issues.pdf Access 2005-04-18

Mindfire solutions, *Porting: A development primer* [WWW-dokument] URL http://www.mindfiresolutions.com/download/Porting=DevelopmentTechniques.pdf Mindfire solutions 2001 Access 2005-04-02

Mindfire solutions. *Porting: The Business Imperative* [WWW-dokument] URL http://www.mindfiresolutions.com/download/Porting=TheBusinessImperative.pdf Mindfire solutions 2001 Access 2005-04-02

Modi, Tarak, *The Platform-Blending Concept,* [WWW-dokument] URL www.teknirvana.com/internal_documents/PlatformBlending.pdf, 2000 Access 2005-04-07

Myers, M. D., *Investigation information systems with ethnographic research*. Auckland: University of Auckland, 1999

Myers, Michael, *References on grounded theory* [WWW-dokument] URL http://www.qual.auckland.ac.nz/grndrefs.htm 2004 Access 2005-04-11

Office of Information Technology, *Mac: File system,* 2002 [WWW-dokument] URL http://www.helpdesk.umd.edu/documents/2/2593/ Access 2005-04-27

Prashant, Jain och Schmidt, Douglas C, *Experiences Converting a C++ Communication Software Framework to Java*, 1997 [WWW-dokument] URL http://www.cs.wustl.edu/~pjain/java/java_notes.html Access 2005-04-12

Quark.com, *tech-notes Cross-Platform Issues: graphics*, 2005 [WWW-dokument] URL http://www.quark.com/service/desktop/support/techinfo/technotes.jsp?idx=36 Access 2005-04-16

Runrev.com, *OT: Mac OS X have a registry?* [WWW-dokument] URL http://lists.runrev.com/pipermail/use-revolution/2003-October/024694.html Access 2005-05-15

Ryall, Ken, *Writing Java on the Mac for All Platforms*, 2004 [WWW-dokument] URL http://www.mactech.com/articles/mactech/Vol.14/14.05/WritingJavaCross-Platform/ Access 2005-05-13

SearchWin2000.com, *Dynamic Link Library* [WWW-dokument] URL http://searchwin2000.techtarget.com/sDefinition/0,sid1_gci213902,00.html Access 2005-04-28

Shields, Ian, *Linux SEK 2005 Release 1: Basic tasks for new Linux developers*, 2005 [WWW-dokument] URL http://www-128.ibm.com/developerworks/linux/library/l-sek51-basics/?ca=dgr-lnxw16NewLinux Access 2005-05-02

Skansholm, Jan, *C++ Direkt,* 2:a upplagan, Lund, Studentlitteratur, 2000

Skansholm, Jan, *Java direkt,* 2:a upplagan, Lund: Studentlitteratur, 1999

Smith, David Mitchell och Robin Simpson. *A Look at Alternatives to Microsoft* [WWW-dokument]. URL http://www.gartner.com/resources/114500/114565/114565.pdf Gartner 2003. Access 2005-04-01

Sohlberg, Peter och Britt-Marie Sohlberg, *Kunskapens former – Vetenskapsteori och forskningsmetod*. Falköping: Liber AB, 2001

Sommerville Ian, Tom Rodden, Pete Sawyer, Richard Bentley och Michael Twidale. *Integrating ethnography into the requirements engineering process*. Lancaster: Lancaster University, 1994

Soulié, Juan, *The cplusplus.com language tutorial,*2003 [WWW-dokument] URL http://www.cplusplus.com/doc/tutorial/ Access 2005-04-16

Sourceforge, *Project: Crypto++: Summary* [WWW-dokument] URL http://sourceforge.net/projects/cryptopp/ Access 2005-04-16

Strauss, Anselm och Juliet Corbin, *Basic qualitative research: Techniques and procedures for developing grounded theory*. 2:a upplagan. USA: Sage Publications, 1998

Sun Microsystems, *Java 2 Platform Standard Edition 5.0 Ap*i, 2004, http://java.sun.com/j2se/1.5.0/docs/api/java/io/File.html Access 2005-05-02

Sun Microsystems, *Java Cryptography Extension (JCE)* [WWW-dokument] URL http://java.sun.com/products/jce/index-14.html Access 2005-05-09

Sun Microsystems, *Java Look and Feel Design Guidelines* [WWW-dokument]. URL http://java.sun.com/products/jlf/ed2/book/index.html Access 2005-04-12

Sun Microsystems, *Java Runtime Environment* [WWW-dokument] URL http://java.sun.com/j2se/desktopjava/jre/index.jsp Access 2005-04-09

Sun Microsystems, *New to Java Programming Center – Unraveling Java Terminology* [WWW-dokument] Access 2005-04-09

Sun Microsystems, *The Java Tutorial*. [WWW-dokument] URL http://java.sun.com/docs/books/tutorial/java/nutsandbolts/bitwise.html Access 2005-04-16

The Shell, *Wine User Guide*, [WWW-dokument] URL http://www.theshell.com/~vinn/wine-user-guide.html Access 2005-04-27

Thurén, Torsten, *Vetenskapsteori för nybörjare*. Malmö: Liber AB, 1991

TIOBE, *Programming Community Index for April 2005*, [WWW-dokument] URL http://www.tiobe.com/tiobe_index/tekst.htm. Access 2005-04-04

Tyma, Paul, *Why are we using Java again?*, 1998 [WWW-dokument] URL http://www.cs.cornell.edu/Ugrad/Tyma_Article.htm Access 2005-04-18

Vanderburg, Glenn L, et al, *Tricks of the Java Programming Gurus*, 1996, [WWW-dokument] URL http://docs.rinet.ru/JaTricks/ch28.htm Access 2005-04-11

Venners, Bill. *Inside the Java Virtual Machine*. McGraw-Hill Companies; 2:a upplagan, 2000

W3Schools, *Browser Statistics,* [WWW-dokument] URL http://www.w3schools.com/browsers/browsers_stats.asp Access 2005-04-20

Wagner, Niel, *The Laws of Cryptography: Unsigned bytes in Java*, 2001 [WWW-Dokument] URL http://www.cs.utsa.edu/~wagner/laws/Abytes.html Access 2005-05-14

Webopedia.com, *Online Computer Dictionary for Computer and Internet Terms and Definitions*. [WWW-dokument] URL http://www.webopedia.com/ Access 2005-04-17

Whatis.com, *Reverse engineering* [WWW-dokument] URL http://whatis.techtarget.com/definition/0,289893,sid9_gci507015,00.html Access 2005-04-14

Wikipedia the Free Encyclopedia. *Library* [WWW-dokument] URL
http://en.wikipedia.org/wiki/Shared_library Access: 2005-04-28

Wikipedia the Free Encyclopedia. *List of web browsers* [WWW-dokument] URL
http://en.wikipedia.org/wiki/List_of_web_browsers Page Access: 2005-04-28

Wikipedia the Free Encyclopedia. *Microsoft Foundation Classes* [WWW-dokument] URL
http://en.wikipedia.org/wiki/Web_browser Access: 2005-04-28

Wikipedia the Free Encyclopedia. *Web browser* [WWW-dokument] URL
http://en.wikipedia.org/wiki/Web_browser Access: 2005-04-28

Winguides, *Windows Registry Tutorial*, [WWW-dokument] URL
http://www.winguides.com/article.php?id=1&page=1&guide=registry Access 2005-04-16

Bilaga 1

Definitioner

Bouncy Castle

Bouncy castle är ett krypterings-API främst utvecklat för Java. API:et består fullt ut av öppen källkod och utvecklas av många personer världen över vilka samlas under namnet The Legion of the Bouncy Castle. API:et innehåller bland annat en mängd olika krypteringsalgoritmer som genom en implementering av Bouncy castle i en Javaapplikation går att nyttja för dess säkerhetskritiska programkod. [265]

C++

Arbetet med att utveckla programspråket C++ påbörjades under 1980 och fick formellt namnet C++ under 1983, då den första manualen publicerades. Boken ”The C++ Programming Language” av grundaren Bjarne Stroustrup, gavs ut i samband med programspråkets första kommersiella utgåva 1985[266]. C++ är ett snabbt och objektorienterat språk. Det är också portabelt, språket kan användas för utveckling för en mängd olika målplattformar, med hjälp av mindre justeringar. Det är också modulärt vilket innebär att det är möjligt att kompilera ett flertal olika källkodsfiler för att sedan länka dem samman. Programspråket är också kompatibelt med sin föregångare C, och C++ användning av speciella tecken gör att källkodens omfattning kan hållas ganska liten. [267]

Crypto++

Crypto++ är ett gratis klassbibliotek för kryptering[268]. Biblioteket gör det möjligt att bland annat skapa och utnyttja nycklar för kryptering och dekryptering. Crypto++ kan också användas för att skapa autenticeringskoder och för att dra nytta av hashfunktioner. Biblioteket används genom programspråket C++.[269]

Dynamic Link Library (dll)

Dynamic Link Library eller dll är en samling av applikationskod som kan anropas då till exempel huvudprogrammet behöver mer funktionalitet. dll:er laddas inte in i ramminnet tillsammans med huvudprogrammet utan laddas först då dll:ens funktioner krävs. Vidare så länkas dll:er ihop med programmet som använder dem istället för att kompileras med huvudprogrammet.[270]

HTML

HTML är en förkortning för HyperText Markup Language liknande SGML och är ett plattformsoberoende språk som används för att skapa Webbdokument genom att utnyttja olika så kallade taggar och attribut. Språket innehåller hundratals taggar vilka kan användas till layout för, och formatering av, dokument för webben.[271] Se även XML, SGML.

---

[265] Bouncy Castle, *The Legion of the Bouncy Castle*
[266] Cplusplus.com, *History of C++*
[267] Cplusplus.com, *Description of C++*
[268] Eskimo, *Crypto++ Library 5.2.1*
[269] Sourceforge, *Project: Crypto++: Summary*
[270] SearchWin2000.com, *Dynamic Link Library*
[271] Webopedia.com, *Online Computer Dictionary for Computer and Internet Terms and Definitions*

JCE

The Java Cryptography Extension (JCE) är ett antal paket som bland annat ger möjlighet att generera krypteringsnycklar, utnyttja algoritmer för meddelandeverifiering (MAC - Message Authentication Code) och använda sig av chiffer. JCE var tidigare ett paket som kunde inkluderas i Java 2 SDK, men är nu integrerad i den. JCE är också kompatibelt med andra, externa krypteringsbibliotek, såsom Bouncy Castle.[272]

Mest/minst signifikanta biten eller byten

Den mest signifikanta biten är den biten i ett binärt nummer som har det högsta värdet. Enklast är att förklara det är genom att beskriva ett decimaltal och dess mest signifikanta bit. I decimaltalet 1000 är det det tal längst till vänster, alltså 1:an som är det mest signifikanta talet vilket beror på att det är det talet som är störst. I det binära talet 0111 är det också det tal längst till vänster som är det mest signifikanta, alltså 0:an.[273]

Den mest signifikanta byten är den byte i ett tal som består av flera byte som har det största värdet. Precis som med bitar är det oftast den byte som är längst till vänster som är den byte som är störst och därmed mest signifikant.[274]

Den minst signifikanta biten är oftast den bit och byten som är längst till höger. Detta gäller i ett big-endian-system, i ett little-endian-system fungerar det tvärtom.[275]

Microsoft Foundation Class Library (MFC)

Microsoft Foundation Class Library (MFC) är ett Microsoftbibliotek som introducerades 1992 i samband med Microsofts C/C++ 7.0 kompilator. MFC har inspirerats av Think Class Library (TCL) för Macintosh och är en del i Microsofts strävan efter att vinna andelar av marknaden för utvecklingsverktyg. Biblioteket samlar delar av Windows API i C++klasser som formar ett ramverk för applikationer med vilket man kan hantera objekt och fördefinierade fönster och vanliga grafiska komponenter.[276]

Reverse engineering

Reverse engineering innebär att ta isär något och se hur det fungerar, för att till exempel kunna förbättra produkten. Idag används reverse engineering ofta inom informationsteknologin för att bryta ner mjukvara och undersöka dess egenskaper och komponeneter. Det kan handla om att använda reverse engineering för att få fram källkod från maskinkod,[277] men även för att från källkod få fram en övergripande struktur över dess uppbyggnad. UML-verktyg kan utifrån källkodsfilerna identifiera klasser, beroenden och sätta ihop dessa till ett strukturellt översiktligt UML-diagram.[278]

SGML

SGML står för Standard Generalized Markup Language. SGML är ett system för att organisera och skapa taggar av text. Till skillnad från XML och HTML hanterar inte SGML formatering av dokument utan snarare reglerna för hur taggar skapas. Dessa kan sedan översättas och användas för formatering på olika sätt. SGML används främst för att hantera

[272] Sun Microsystems, *Java Cryptography Extension (JCE)*
[273] Webopedia.com, *Online Computer Dictionary for Computer and Internet Terms and Definitions*
[274] Ibid.
[275] Ibid.
[276] Wikipedia, The Free Encyclopedia*, Microsoft Foundation Classes*
[277] Whatis.com *Reverse engineering*
[278] Developer.com *UML Tools*

större dokument som frekvent behöver omarbetas eller skrivas ut i olika format.[279] Se även HTML, XML

Shared Object Library (so)

Shared Object Library eller so är bibliotek som består av småprogram och som används vid utveckling av mjukvara. Ett Shared Object Library skiljer sig från ett vanligt program på det sättet att det inte är ett självständigt program, utan istället programkod som hjälper applikationer med vissa funktioner.[280]

SKG (Synchronous Key Generator)

SKG är en komponent som hanterar säkra nycklar genom att synkront generera identiska 160-bitars-nycklar på fysiskt separerade plaster. Nycklarna som SKG:n generar kan användas för till exempel autentisering och för kryptering. För varje kommunikationskanal och för varje interaktion mellan dessa skapas nya nycklar vilket gör att även om någon lyckats finna nyckeln för till exempel ett hemligt meddelande så kommer den nyckeln att vara obrukbar på alla andra meddelanden som skickats eller kommer att skickas i den kommunikationskanalen. SKG fungerar på Windows, Mac OS, Linux och Unix. SKG utvecklas av Impsys Digital Security AB.[281]

Windows register

Windows register är en databas som används för att lagra information om en PC i form av bland annat inställningar för hårdvara, mjukvara och användare. Man kan utnyttja registret med hjälp av en editor och därigenom lagra och hämta relevant information till programvaran. En stor fördel med registret är möjligheten att importera och exportera registret till textfiler som sedan kan lagras eller delas med andra för att enkelt uppdatera det lokala registret.[282]

XML

XML står för Extensible Markup Language som är ett plattformsoberoende språk som utvecklats av W3C som används för att skapa webbdokument genom att utnyttja så kallade taggar och attribut. XML har utvecklats ur SGML. I XML tillåts programmeraren att definiera sina egna taggar för att lagra och hantera data.[283] Se även HTML, SGML.

[279] Webopedia.com, *Online Computer Dictionary for Computer and Internet Terms and Definitions.*
[280] Wikipedia the Free Encyclopedia. *Library.*
[281] Impsys Digital Security AB, *The Home of Digital Security.*
[282] Winguides, *Windows Registry Tutorial.*
[283] Webopedia.com, *Online Computer Dictionary for Computer and Internet Terms and Definitions.*